

# Designing a datapath

## Digital Systems – Lecture 21



Department of  
COMPUTER  
SCIENCE

# In this part

---

Stage-by-stage development of a *single-cycle* datapath able to execute many Thumb instructions.

- We won't have pipelined fetch and decode – so the clock speed will be limited.
- We won't implement the control needed for multi-cycle instructions like push and pop.
- And no interrupts, either.

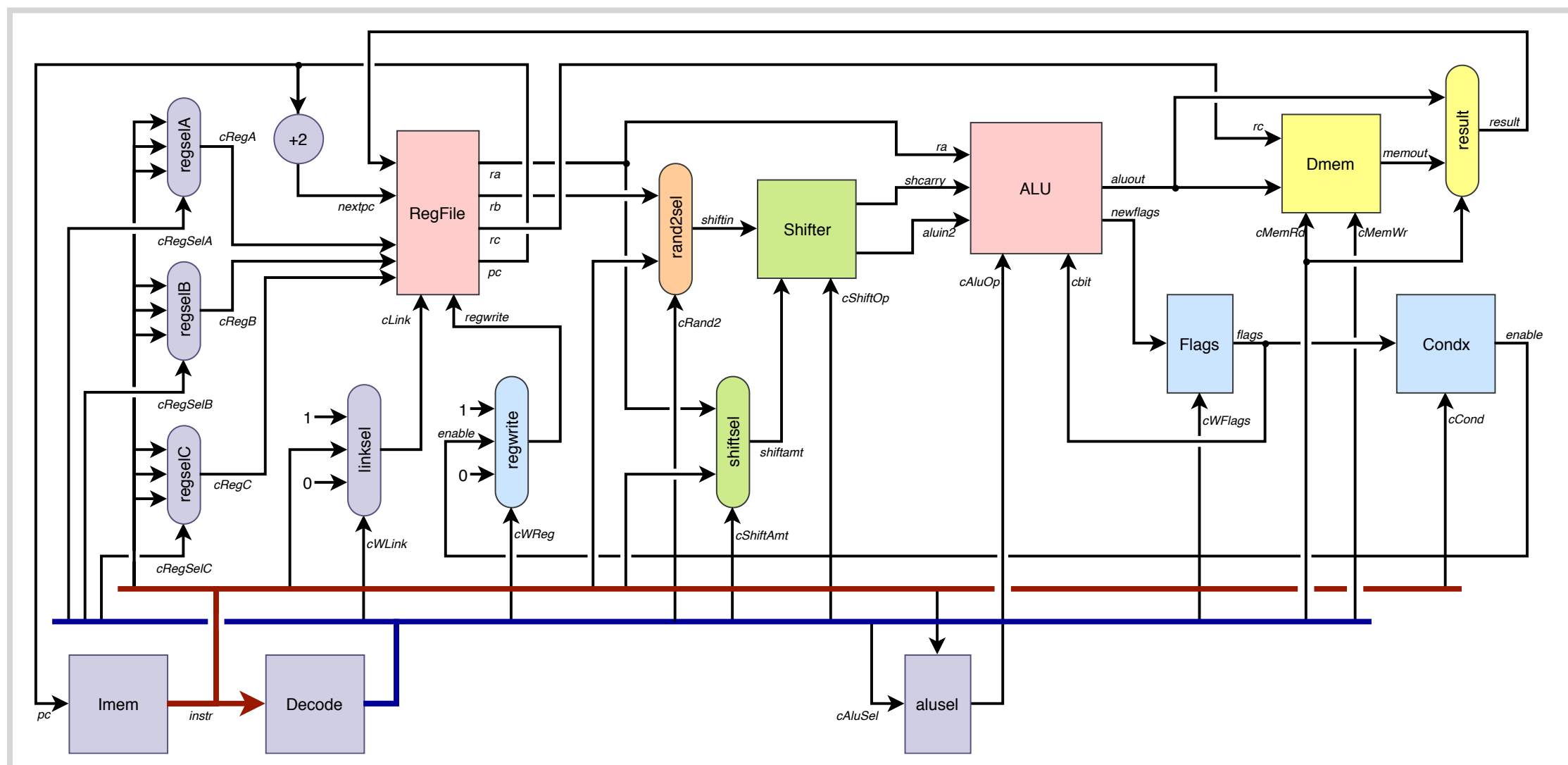
# In this lecture

---

Stage-by-stage development of a datapath design:

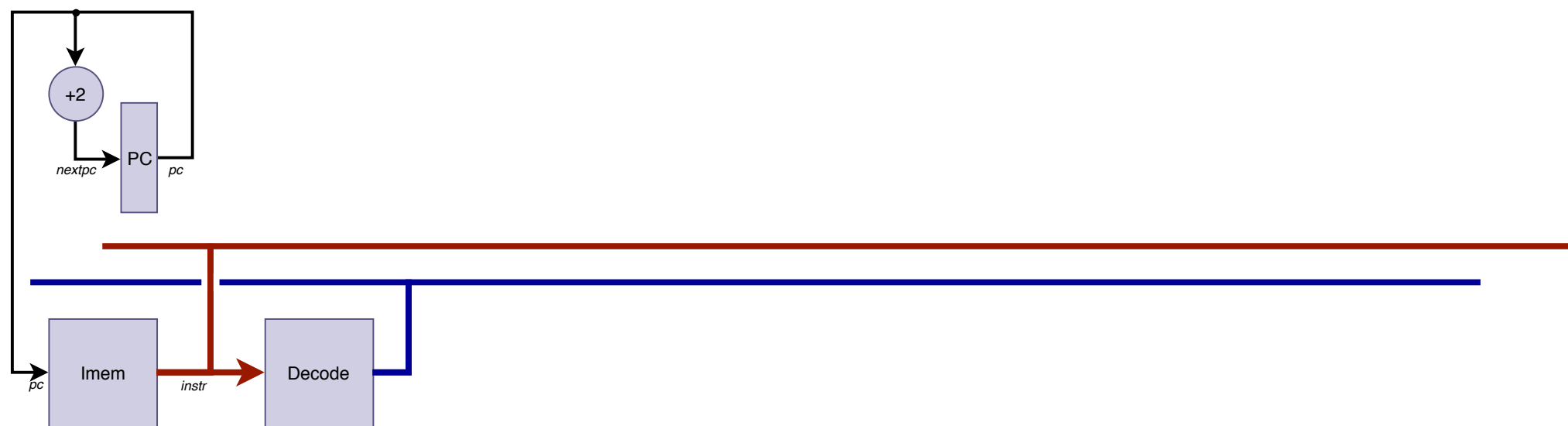
- Fetching and decoding
- Arithmetic between registers
- Immediate operands
- Load and store instructions
- Shifts

# The complete datapath

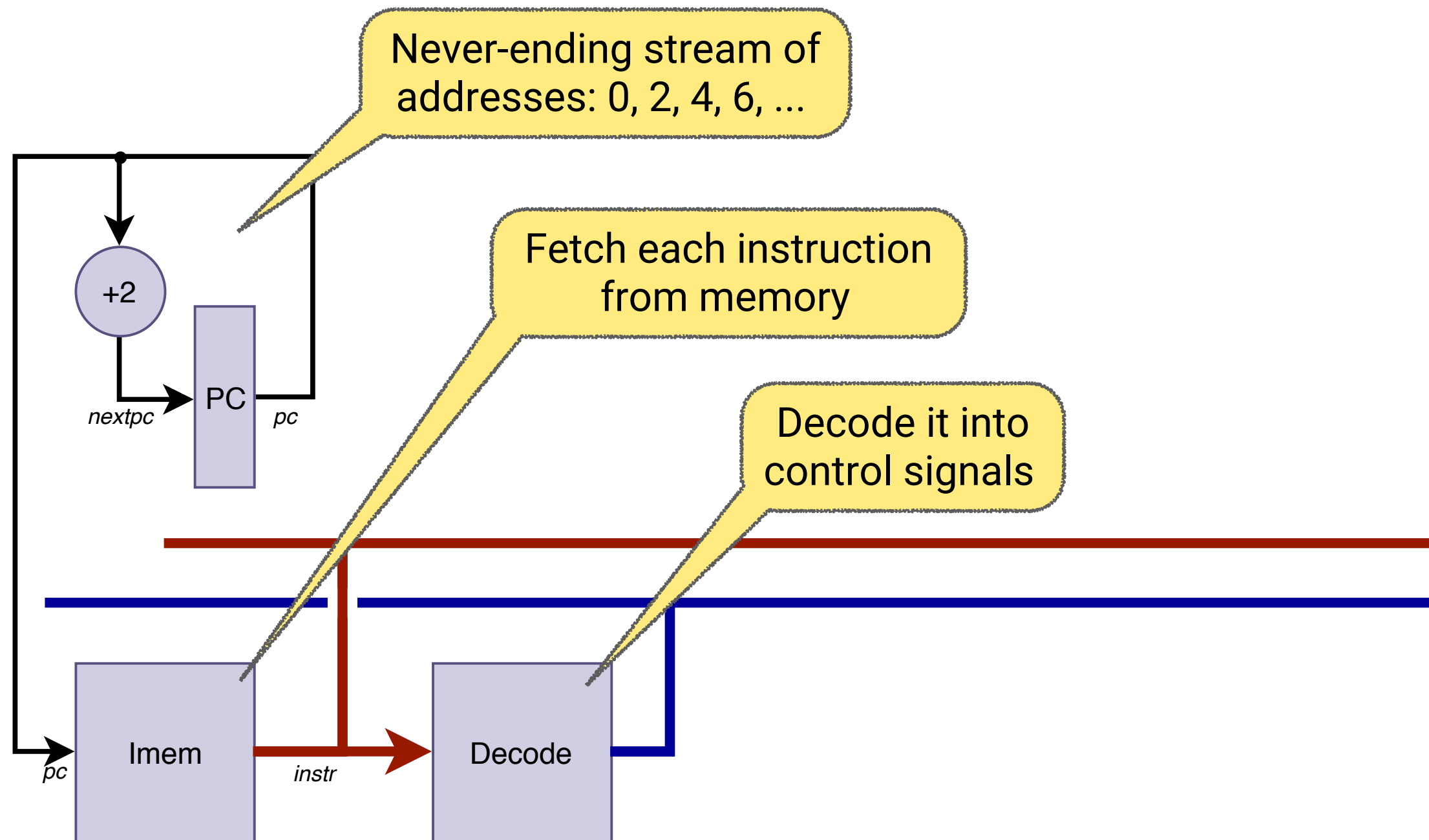




# Stage 1: Instruction fetch



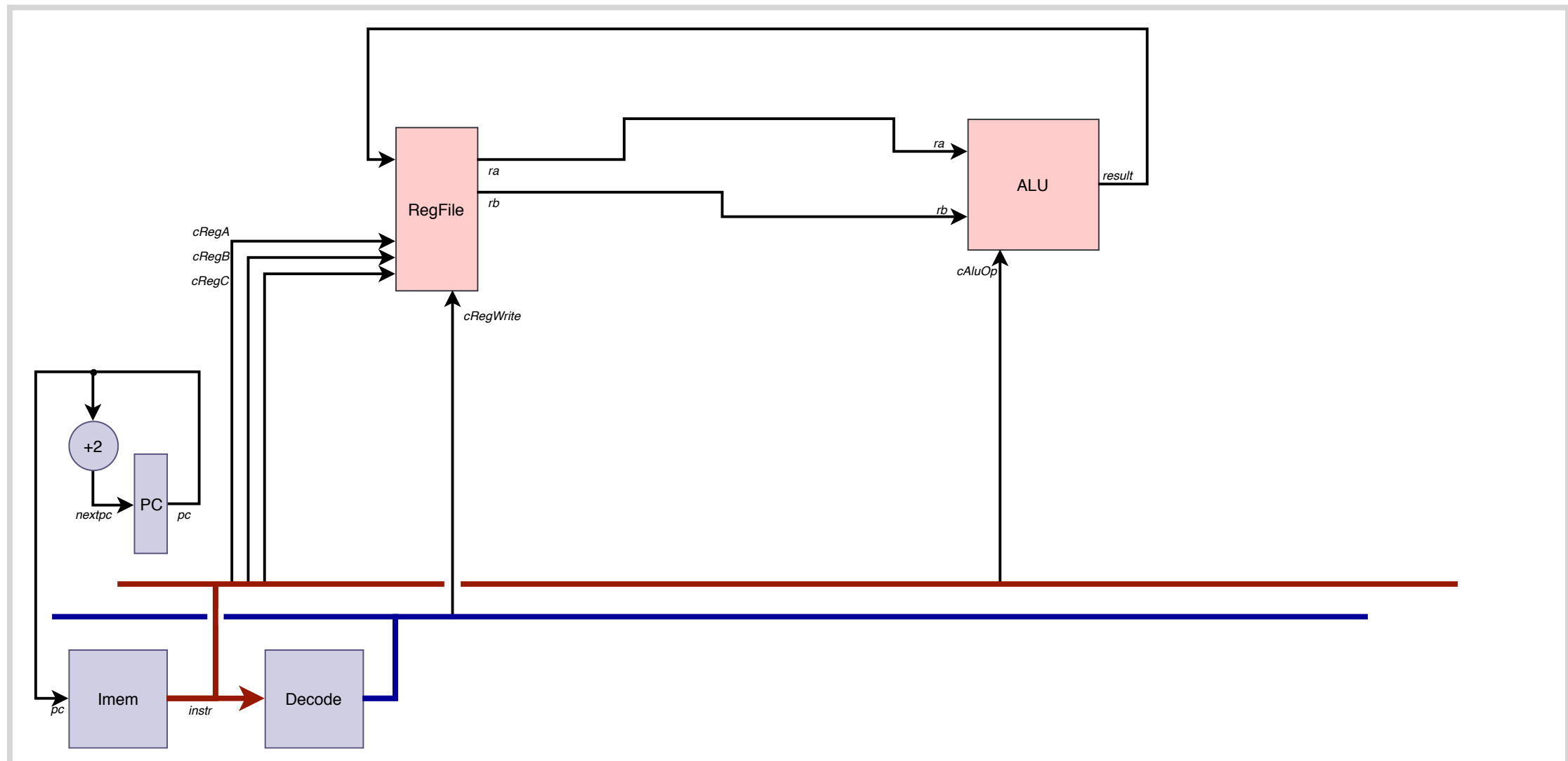
# Stage 1: Instruction fetch



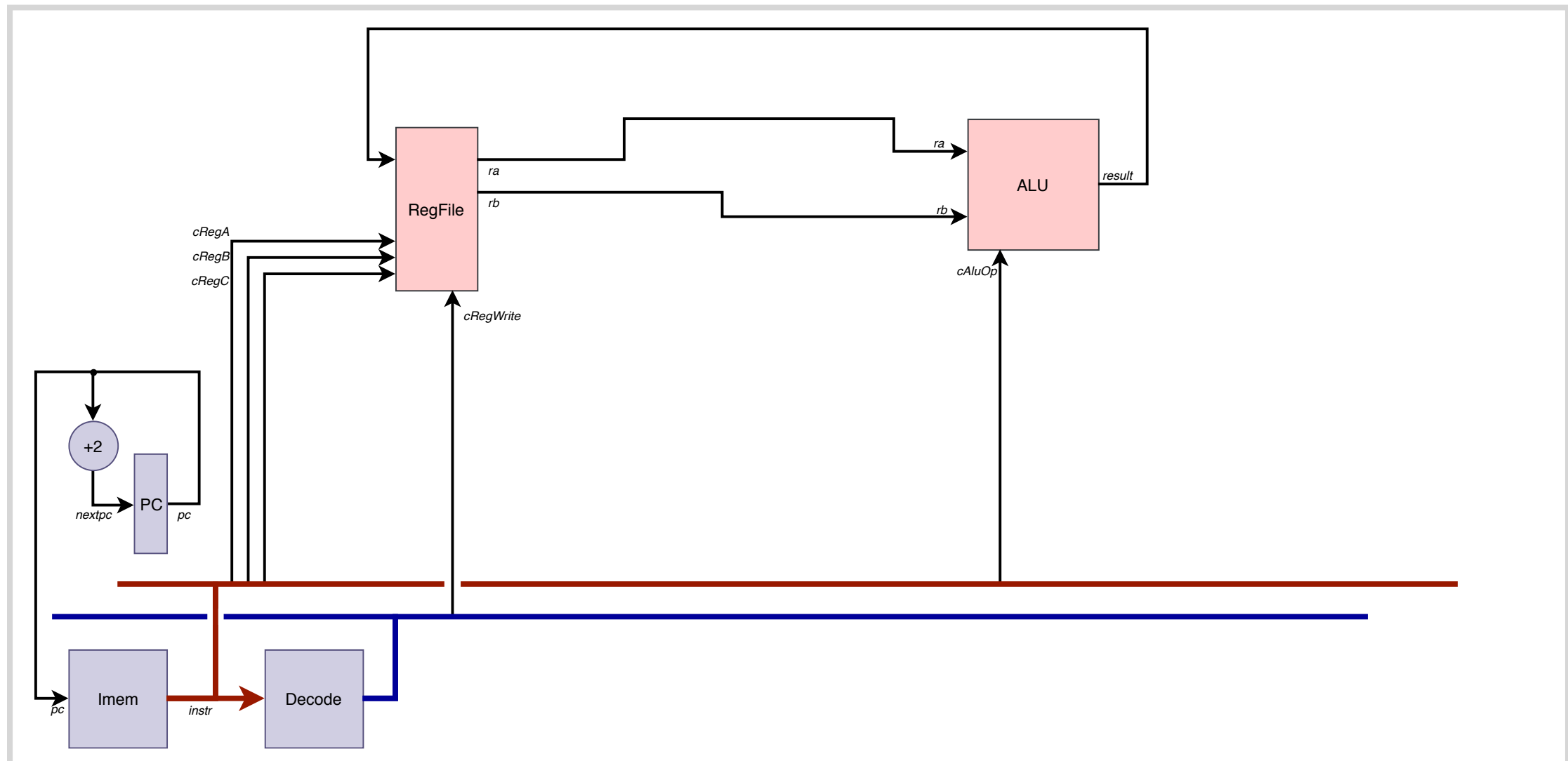
# Stage 1: Instruction fetch



# Stage 2: ALU operations



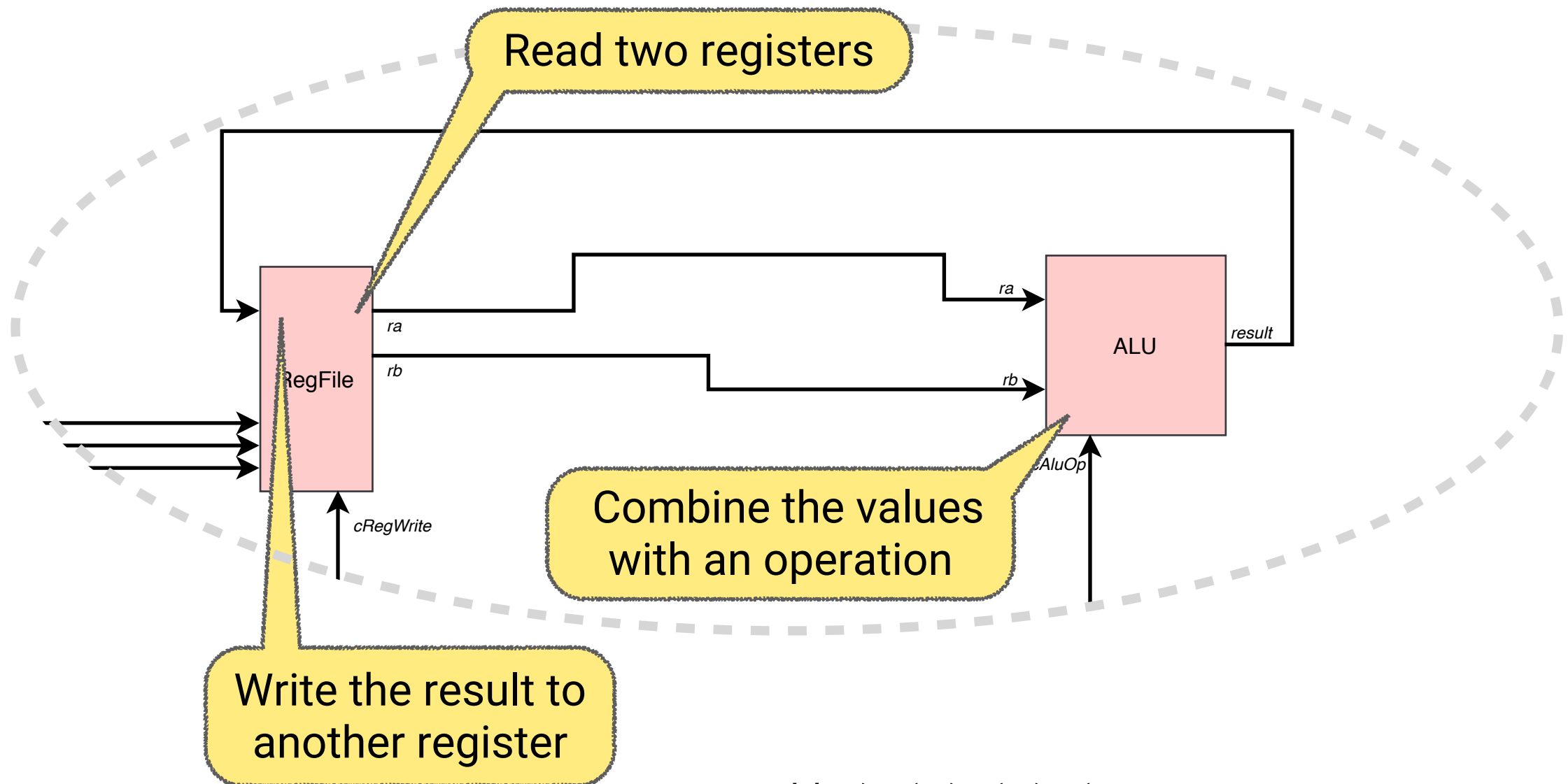
# Stage 2: ALU operations



adds  $\langle Rx \rangle, \langle Ry \rangle, \langle Rz \rangle$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	0	0	Rz			Ry			Rx		

# Stage 2: ALU operations



adds  $\langle Rx \rangle, \langle Ry \rangle, \langle Rz \rangle$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	0	0	Rz			Ry			Rx		

# Reg-to-reg operations

adds  $\langle Rx \rangle, \langle Ry \rangle, \langle Rz \rangle$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	0	0	Rz			Ry			Rx		

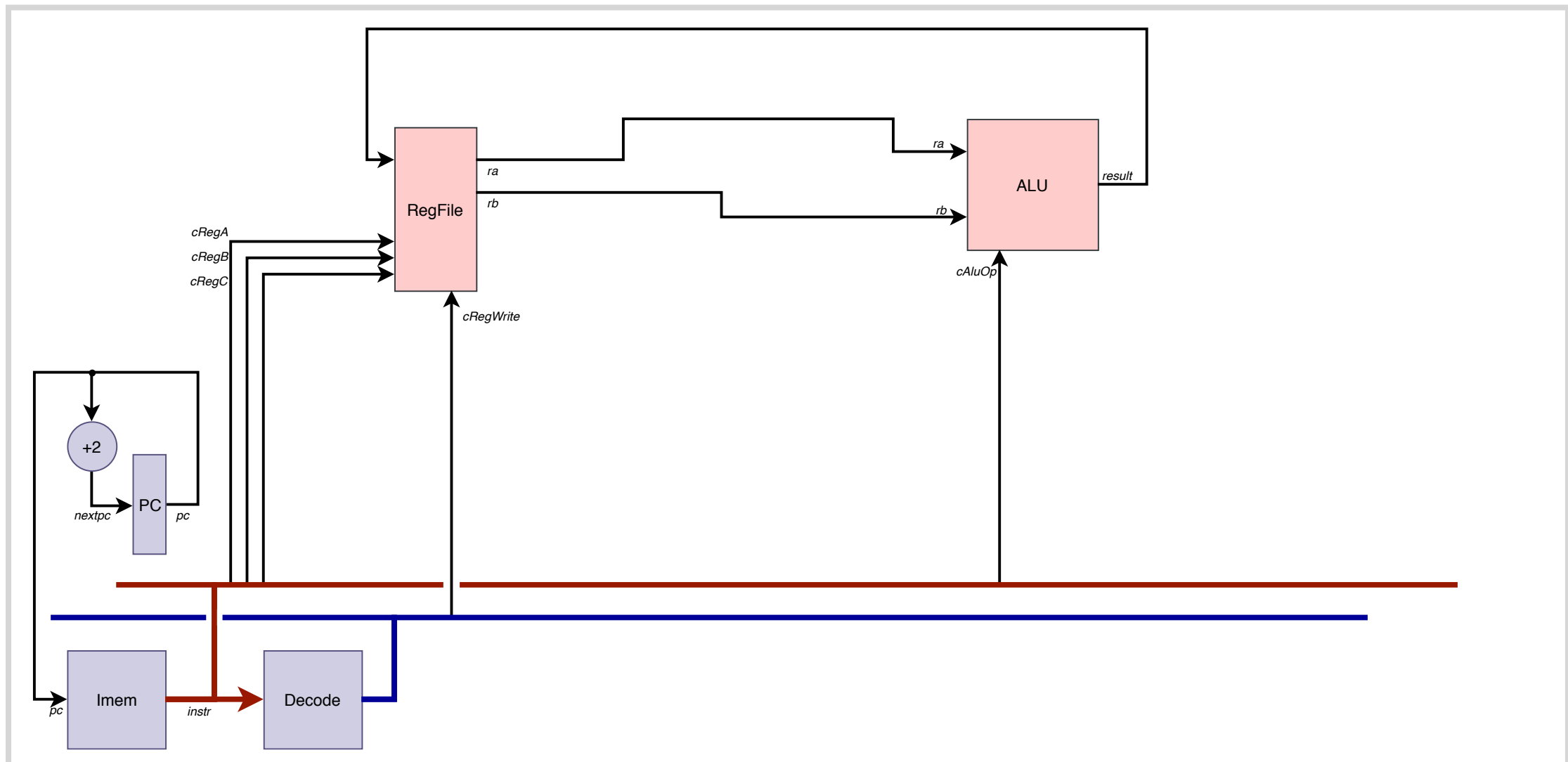
subs  $\langle Rx \rangle, \langle Ry \rangle, \langle Rz \rangle$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	0	1	Rz			Ry			Rx		

ands  $\langle Rx \rangle, \langle Ry \rangle$

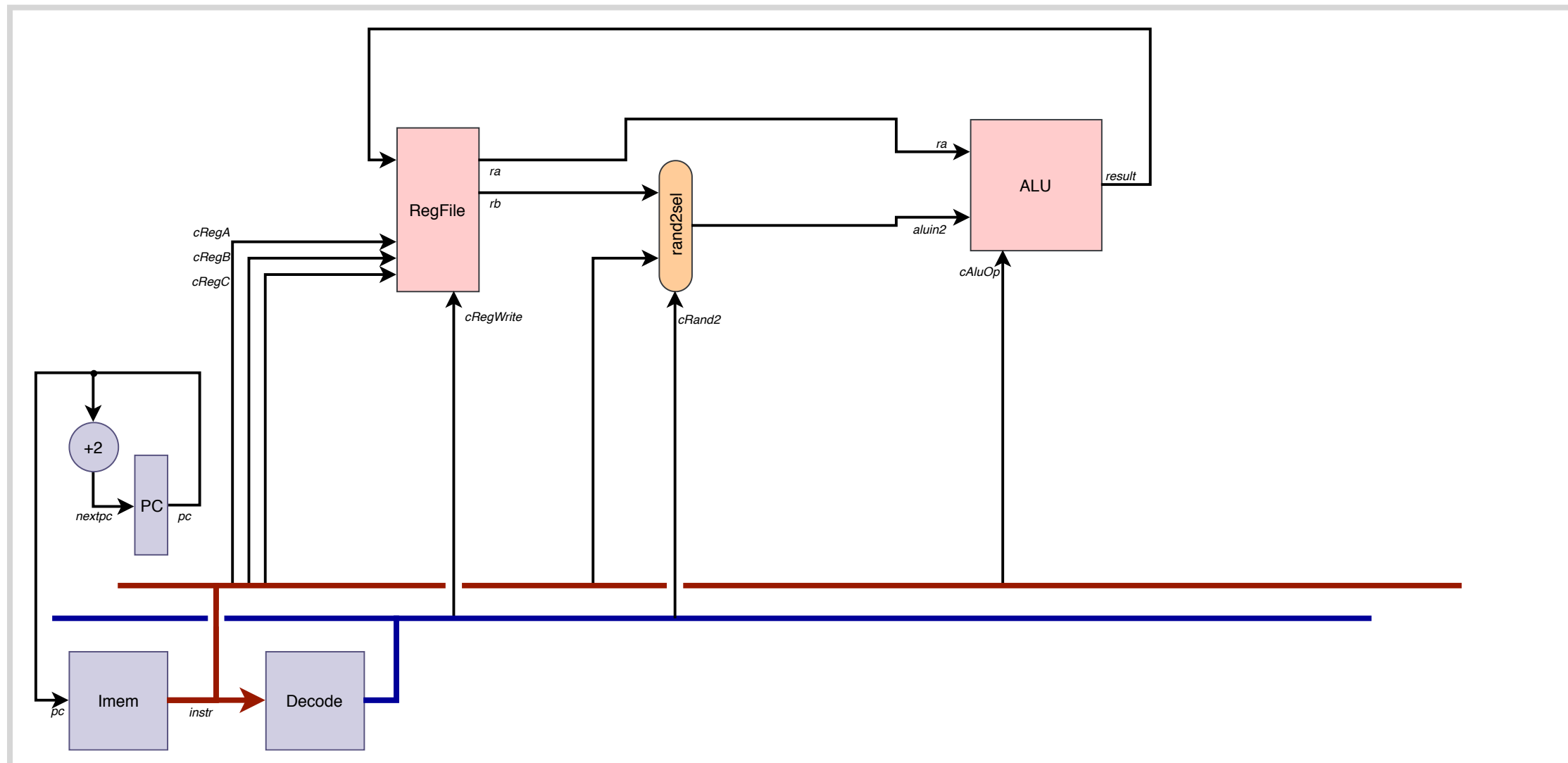
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	0	0	Ry			Rx		

# Stage 2: ALU operations

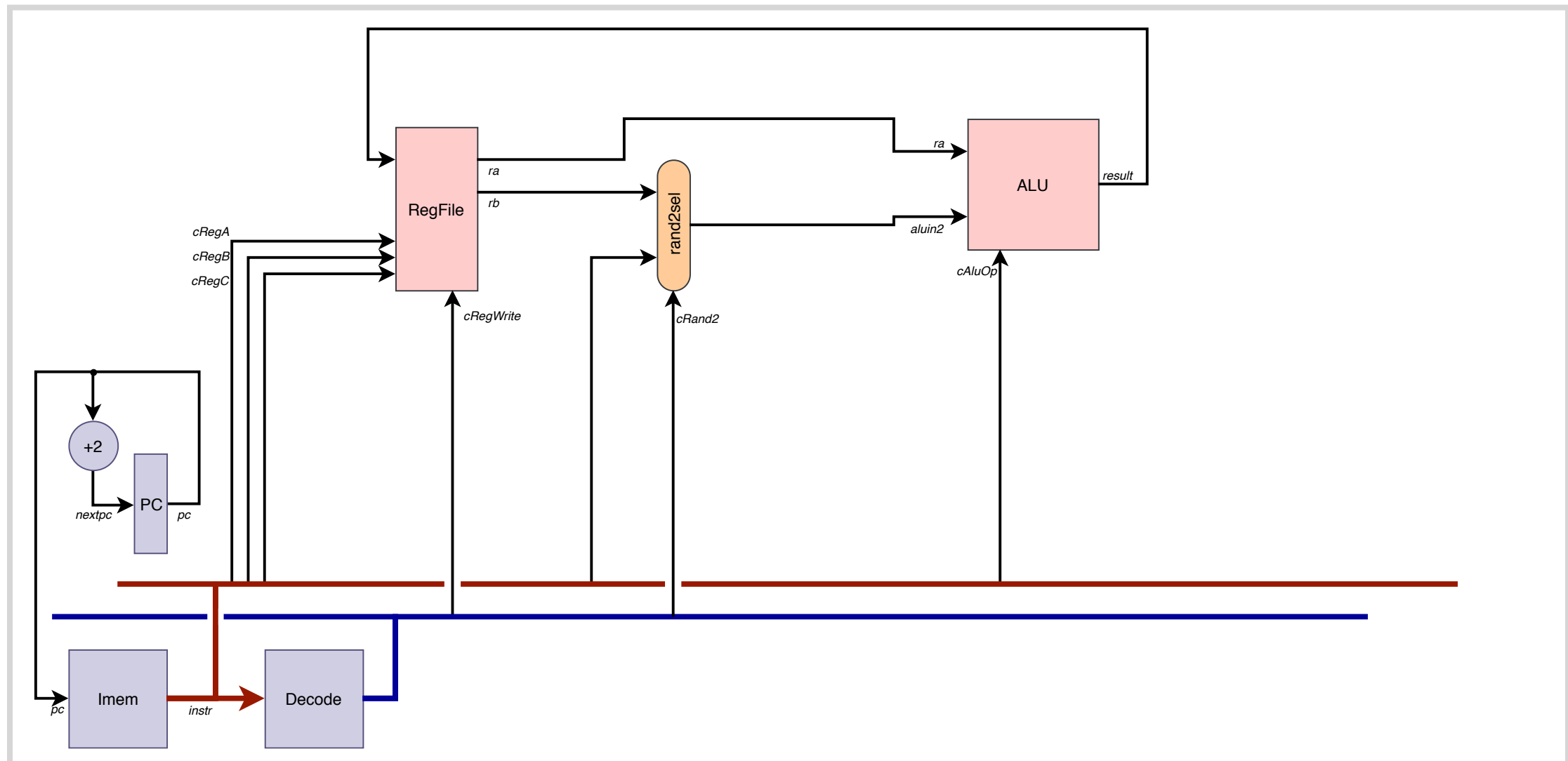




# Stage 3: Immediate operands



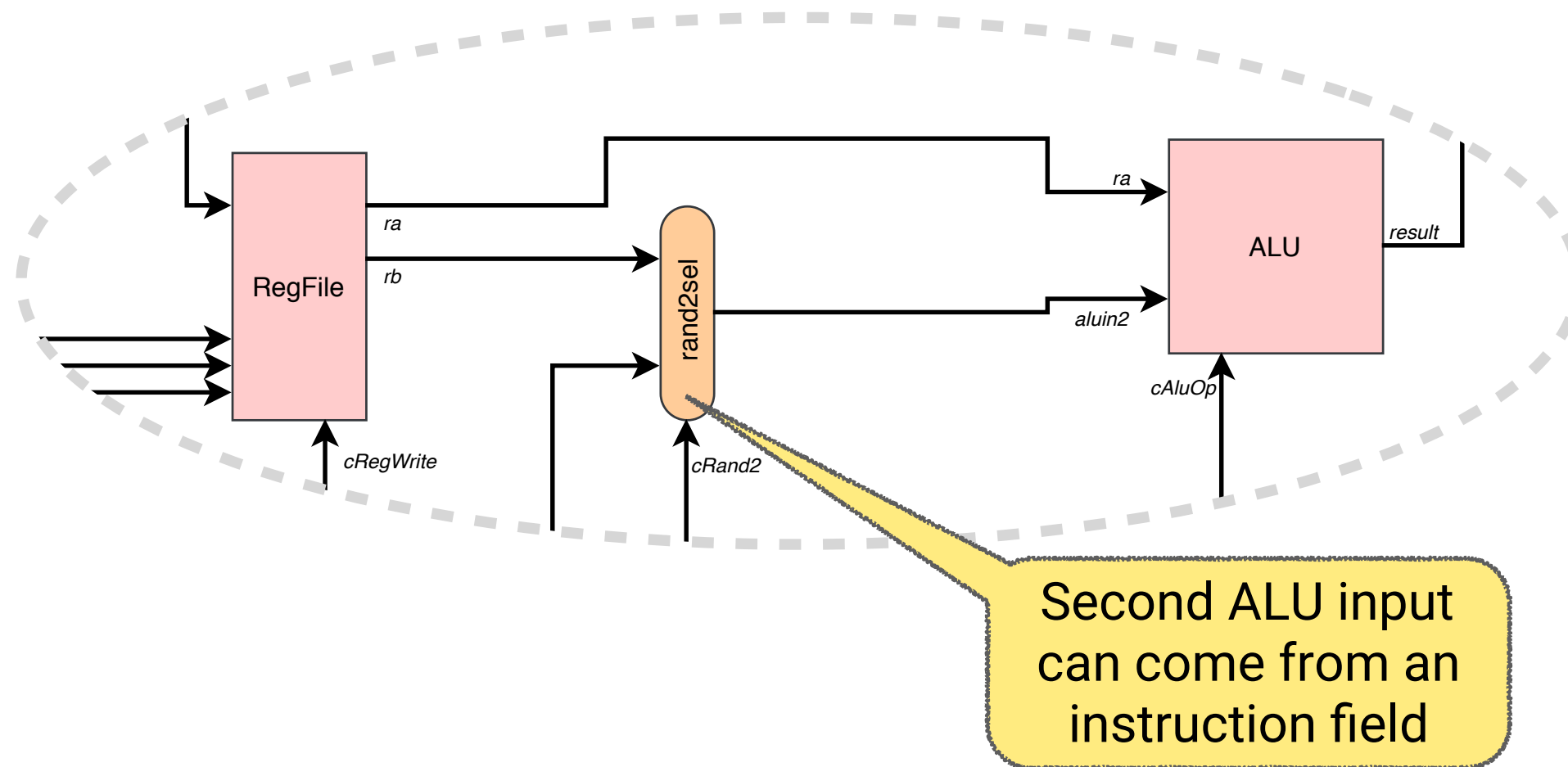
# Stage 3: Immediate operands



adds  $\langle Rx \rangle, \langle Ry \rangle, \# \langle imm3 \rangle$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	1	0	imm3			Ry			Rx		

# Stage 3: Immediate operands



`adds <Rx>,<Ry>,<#imm3>`

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	1	0	imm3			Ry			Rx		

# Operations with immediate field

adds  $\langle Rx \rangle, \langle Ry \rangle, \# \langle imm3 \rangle$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	1	0	imm3			Ry			Rx		

adds  $\langle Rw \rangle, \# \langle imm8 \rangle$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	Rw			imm8							

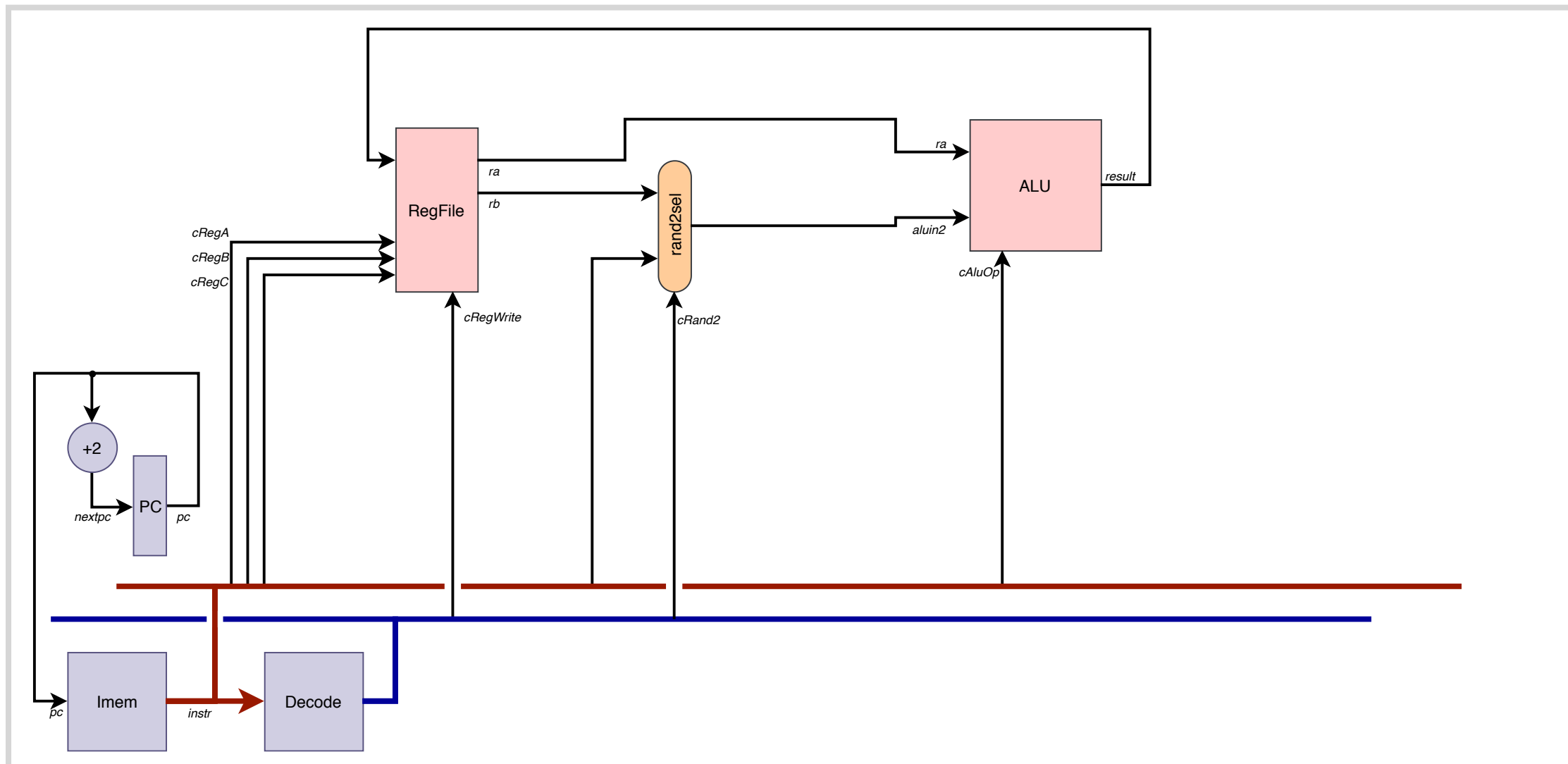
movs  $\langle Rw \rangle, \# \langle imm8 \rangle$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	Rw			imm8							

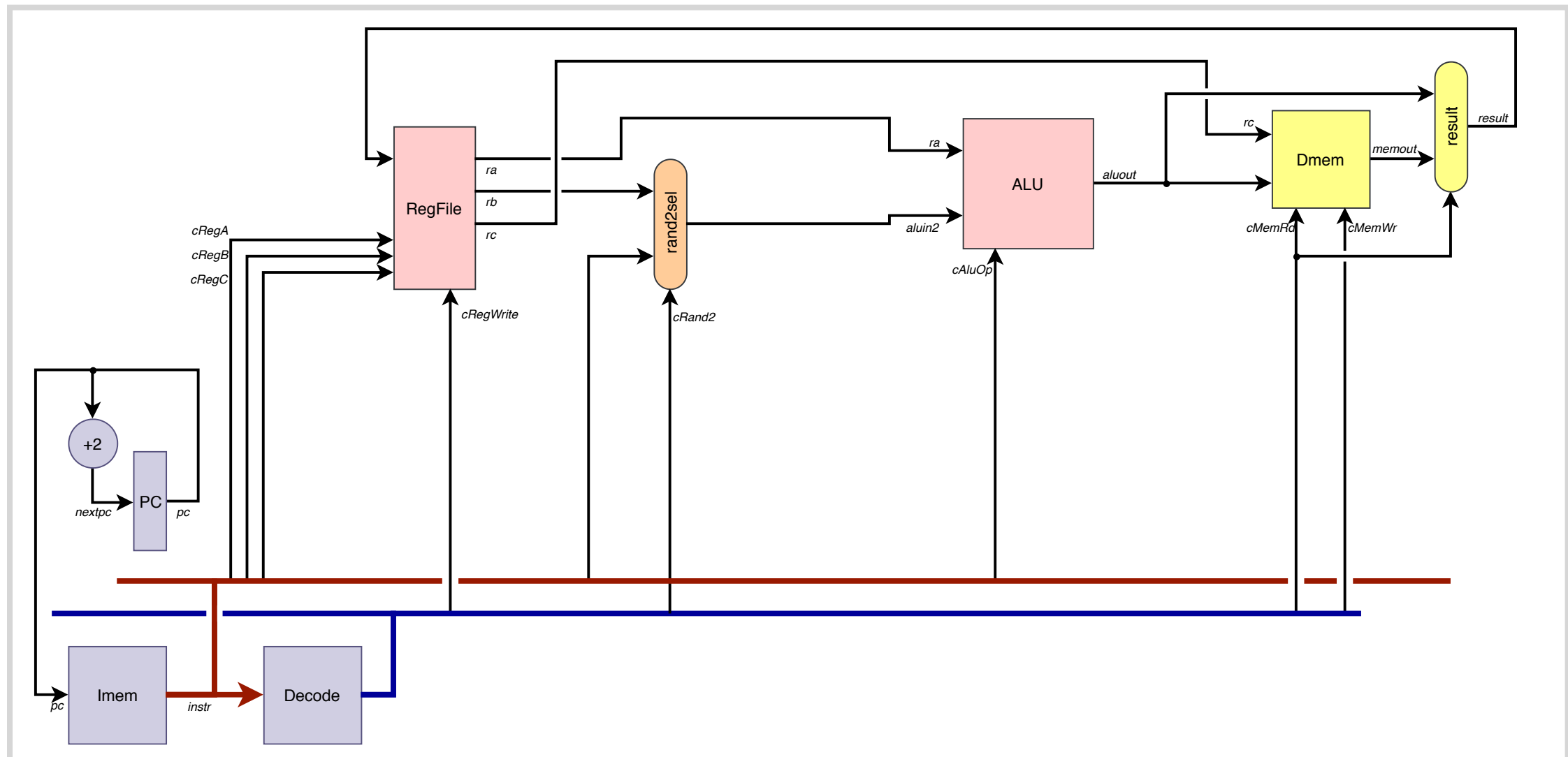
# Control signals

Instruction	cRand2	cAluOp	cRegWrite
adds r	RegB	Add	T
adds i3	Imm3	Add	T
adds i8	Imm8	Add	T
subs r	RegB	Sub	T
subs i3	Imm3	Sub	T
subs i8	Imm8	Sub	T
ands r	RegB	And	T
movs r	RegB	Mov	T
movs i8	Imm8	Mov	T

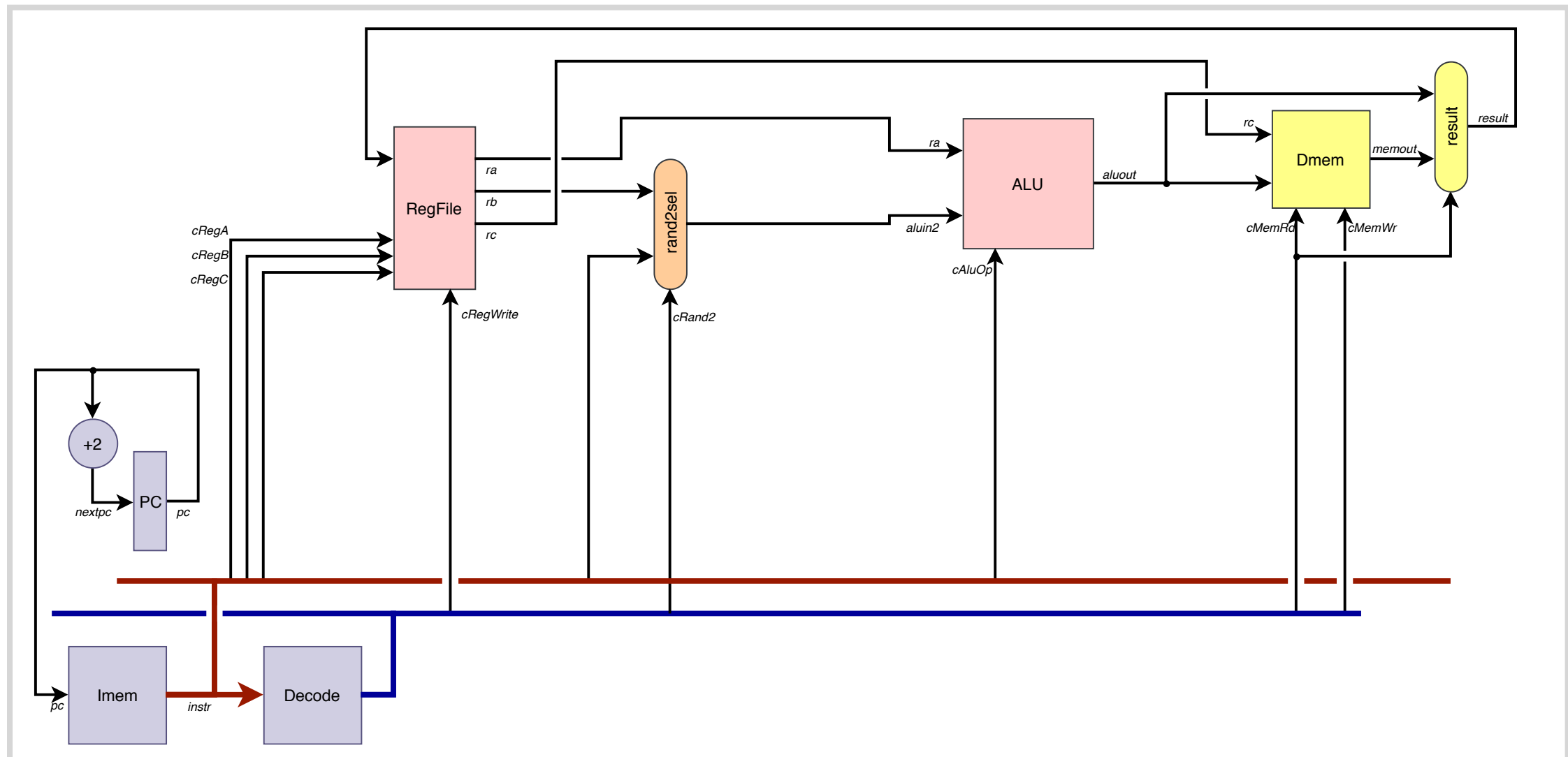
# Stage 3: Immediate operands



# Stage 4: Data memory



# Stage 4: Data memory

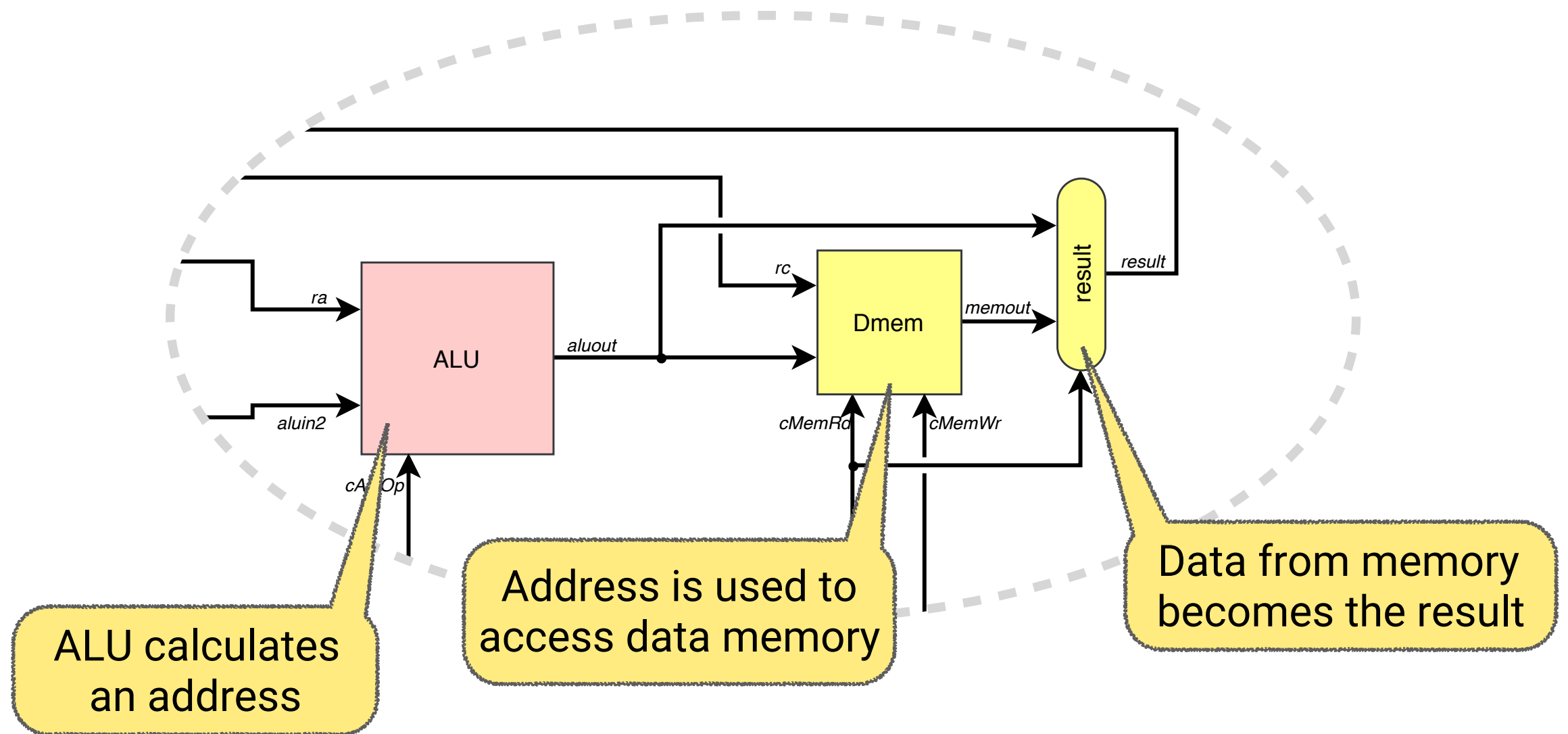


ldr <Rx>,[<Ry>,<Rz>]

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	0	0	Rz			Ry			Rx		



# Stage 4: Data memory



ldr <Rx>,[<Ry>,<Rz>]

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	0	0	Rz			Ry			Rx		

# Load and store instructions

ldr <Rx>,[<Ry>,<Rz>]

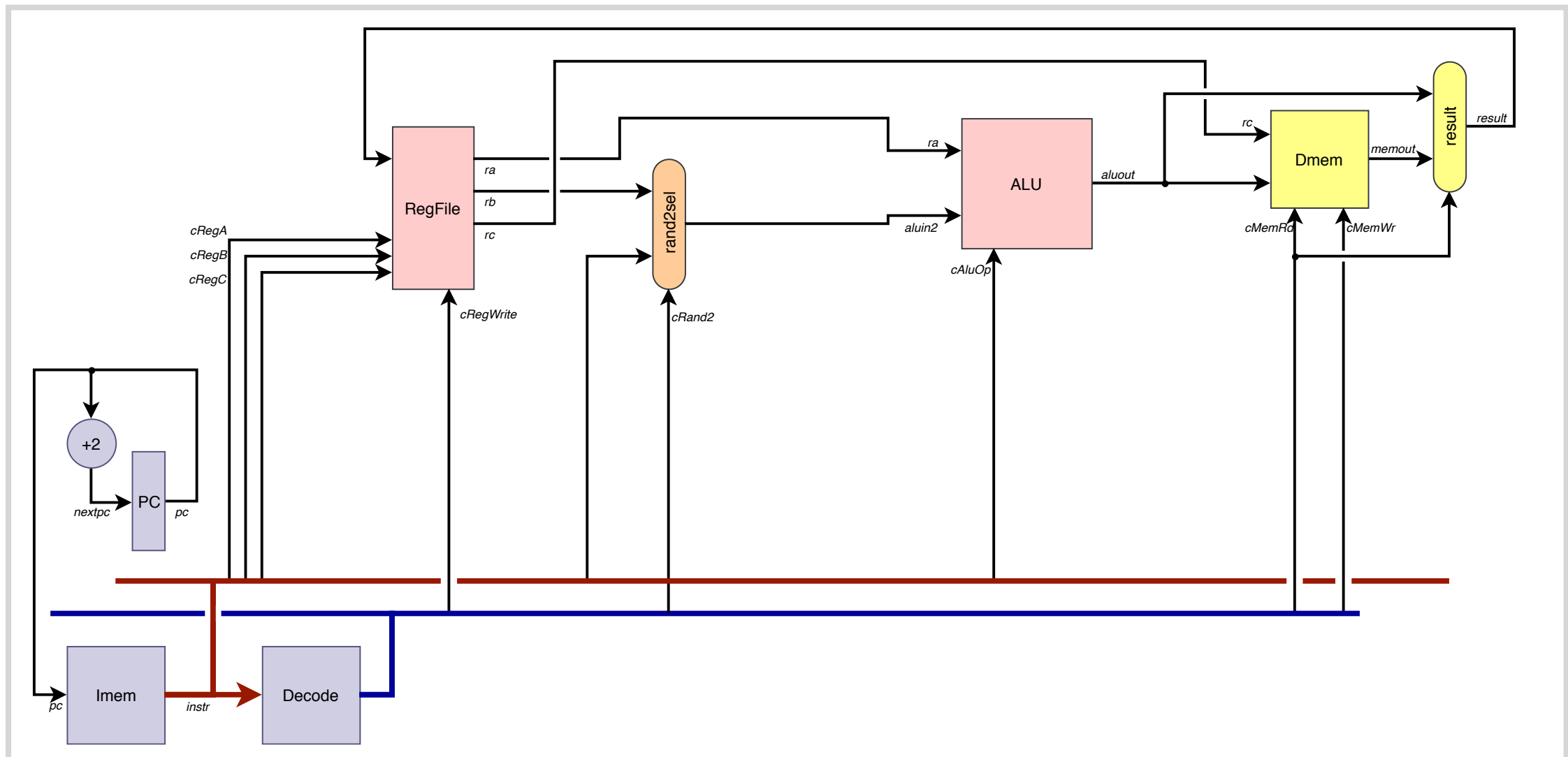
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	0	0	Rz		Ry		Rx				

str <Rx>,[<Ry>,<Rz>]

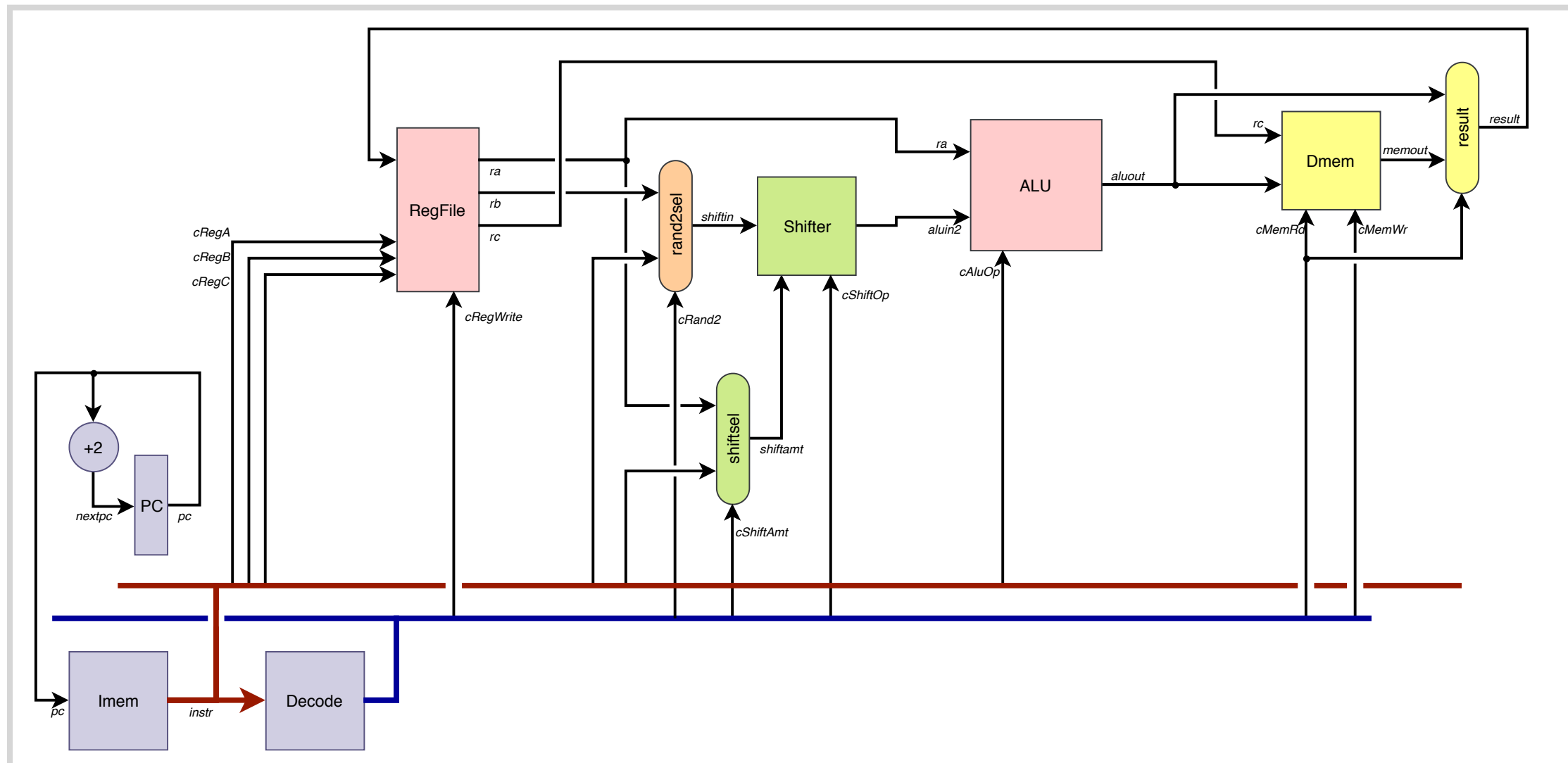
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	0	0	Rz		Ry		Rx				

Instruction	cRand2	cAluOp	cMemRd	cMemWr	cRegWrite
adds r	RegB	Add	F	F	T
movs i8	Imm8	Mov	F	F	T
ldr r	RegB	Add	T	F	T
str r	RegB	Add	F	T	F

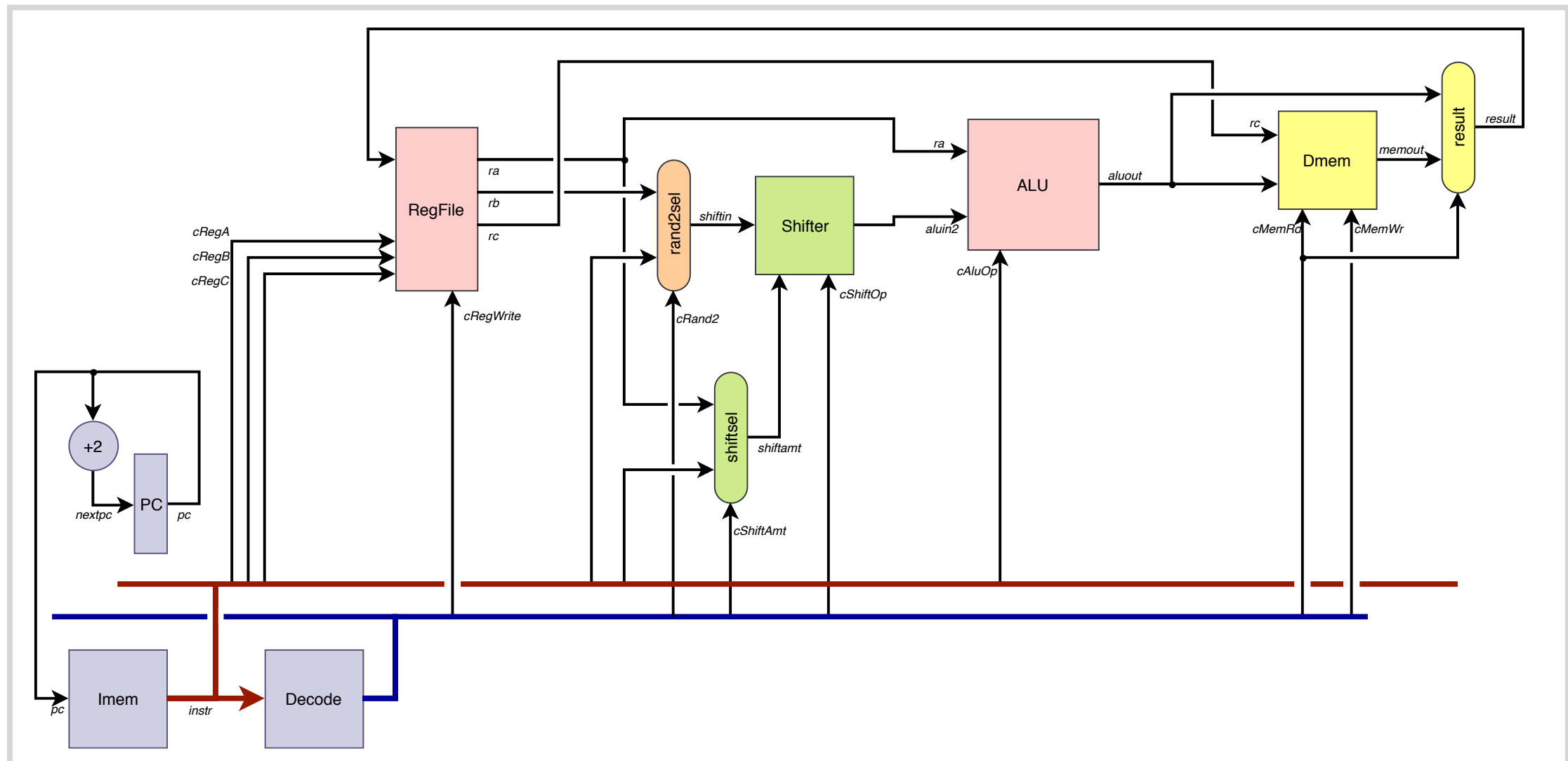
# Stage 4: Data memory



# Stage 5: Barrel shifter



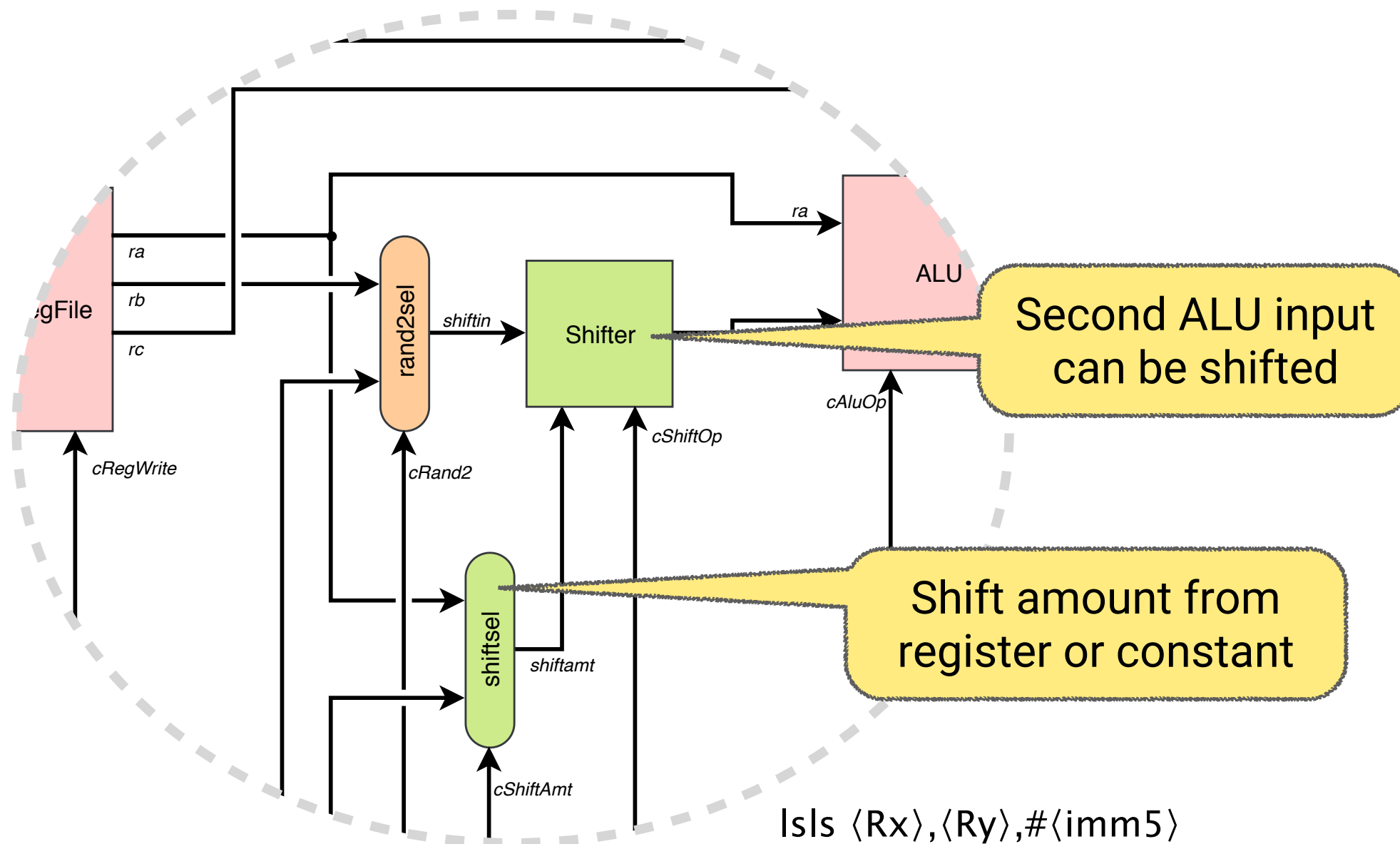
# Stage 5: Barrel shifter



IsIs  $\langle Rx \rangle, \langle Ry \rangle, \# \langle imm5 \rangle$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	imm5					Ry			Rx		

# Stage 5: Barrel shifter



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	imm5					Ry			Rx		

# Shifts and immediate load/stores

lsls  $\langle Rx \rangle, \langle Ry \rangle, \# \langle imm5 \rangle$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	imm5					Ry		Rx			

rors  $\langle Rx \rangle, \langle Ry \rangle$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	1	1	1	Ry		Rx		

ldr  $\langle Rx \rangle, [\langle Ry \rangle, \# \langle imm5 \rangle]$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	imm5					Ry		Rx			

str  $\langle Rx \rangle, [\langle Ry \rangle, \# \langle imm5 \rangle]$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	imm5					Ry		Rx			

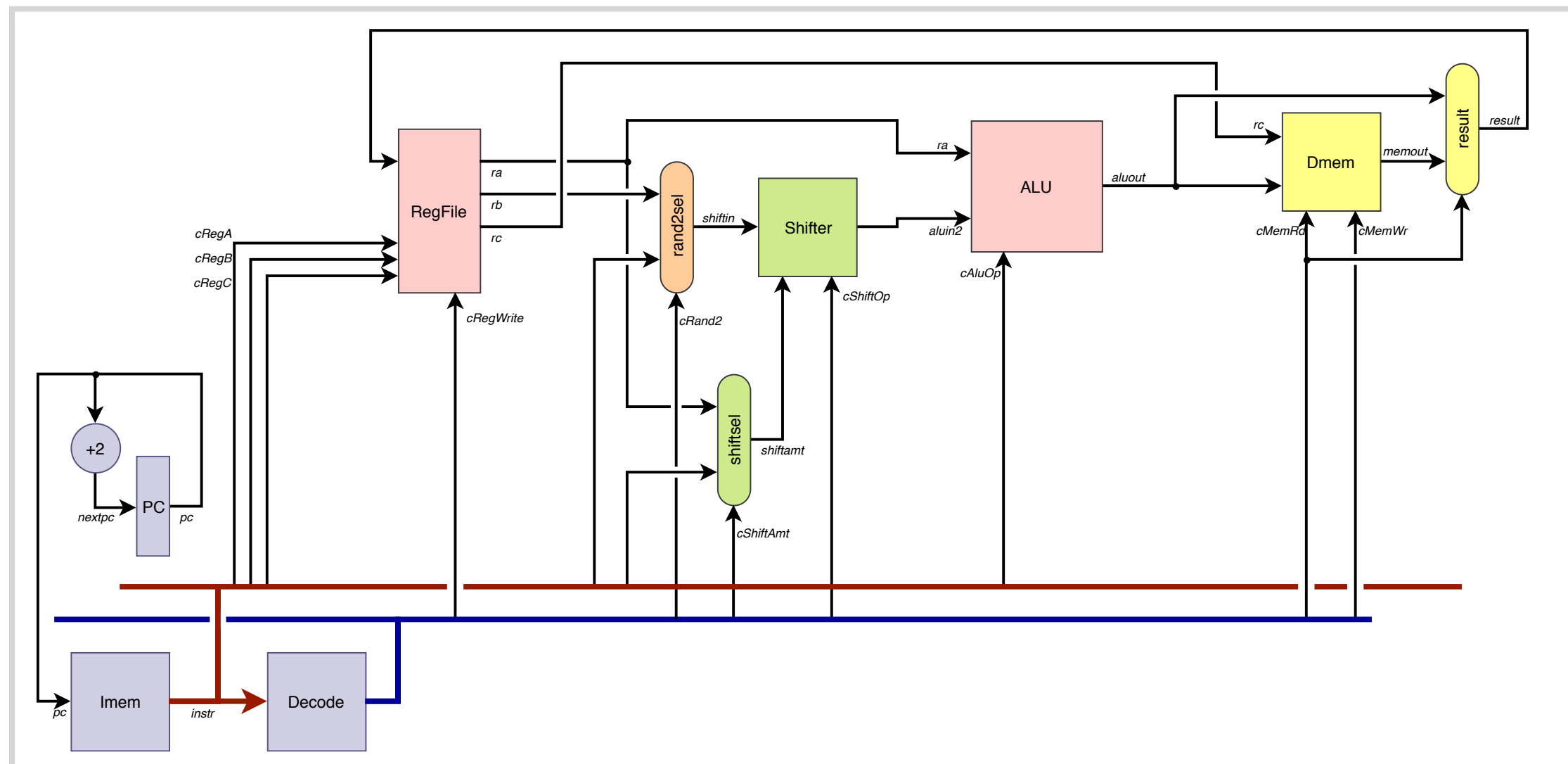


# Updated control signals

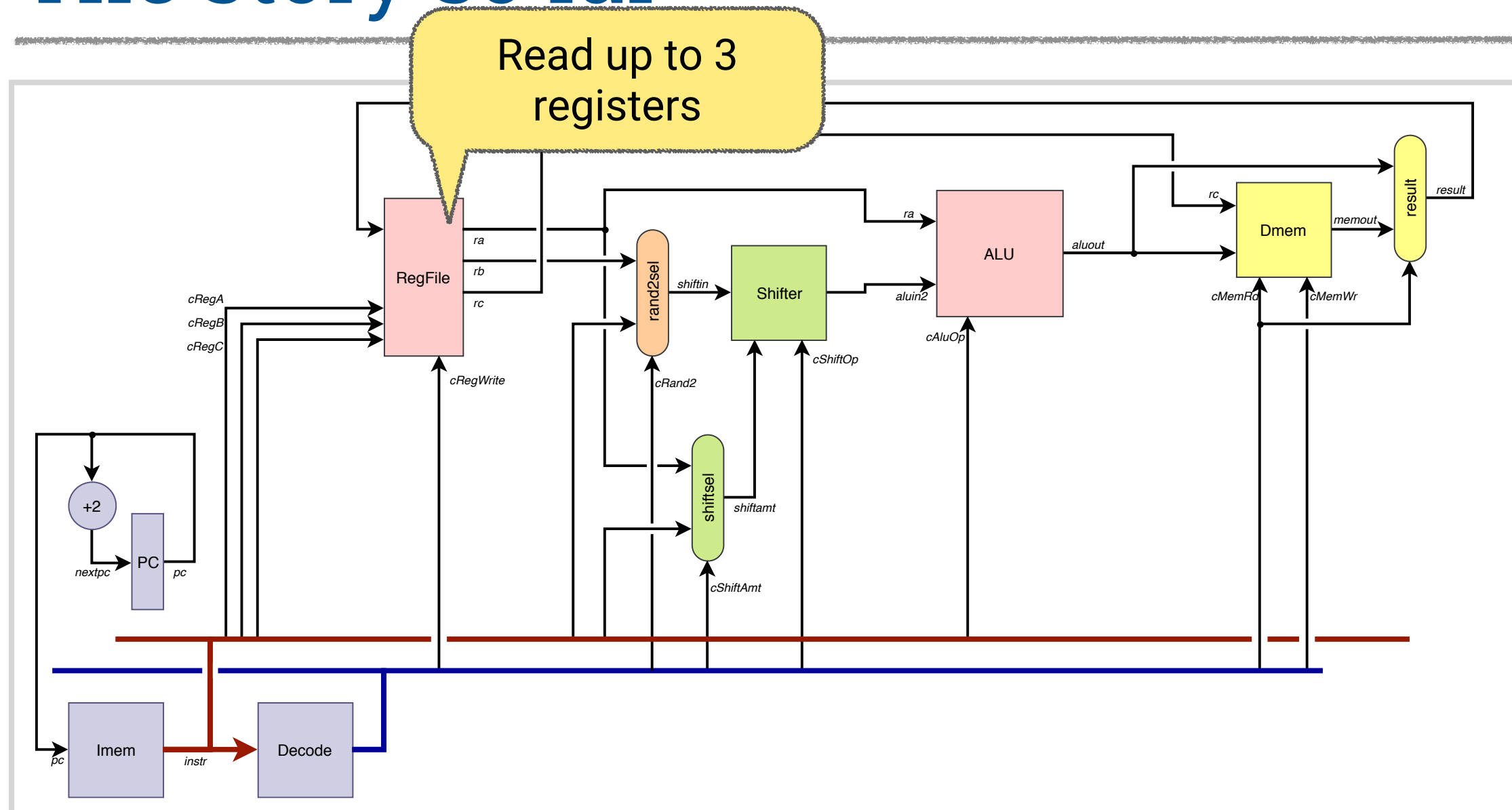
Instruction	cRand2	cShiftOp	cShiftAmt	cAluOp	cMemRd/Wr	cRegWrite
adds r	RegB	Ls1	Sh0	Add	F/F	T
movs i8	Imm8	Ls1	Sh0	Mov	F/F	T
ldr r	RegB	Ls1	Sh0	Add	T/F	T
str r	RegB	Ls1	Sh0	Add	F/T	F
lsls i5	RegB	Ls1	ShImm	Mov	F/F	T
rors r	RegB	Ror	ShReg	Mov	F/F	T
ldr i5	Imm5	Ls1	Sh2	Add	T/F	T
str i5	Imm5	Ls1	Sh2	Add	F/T	F



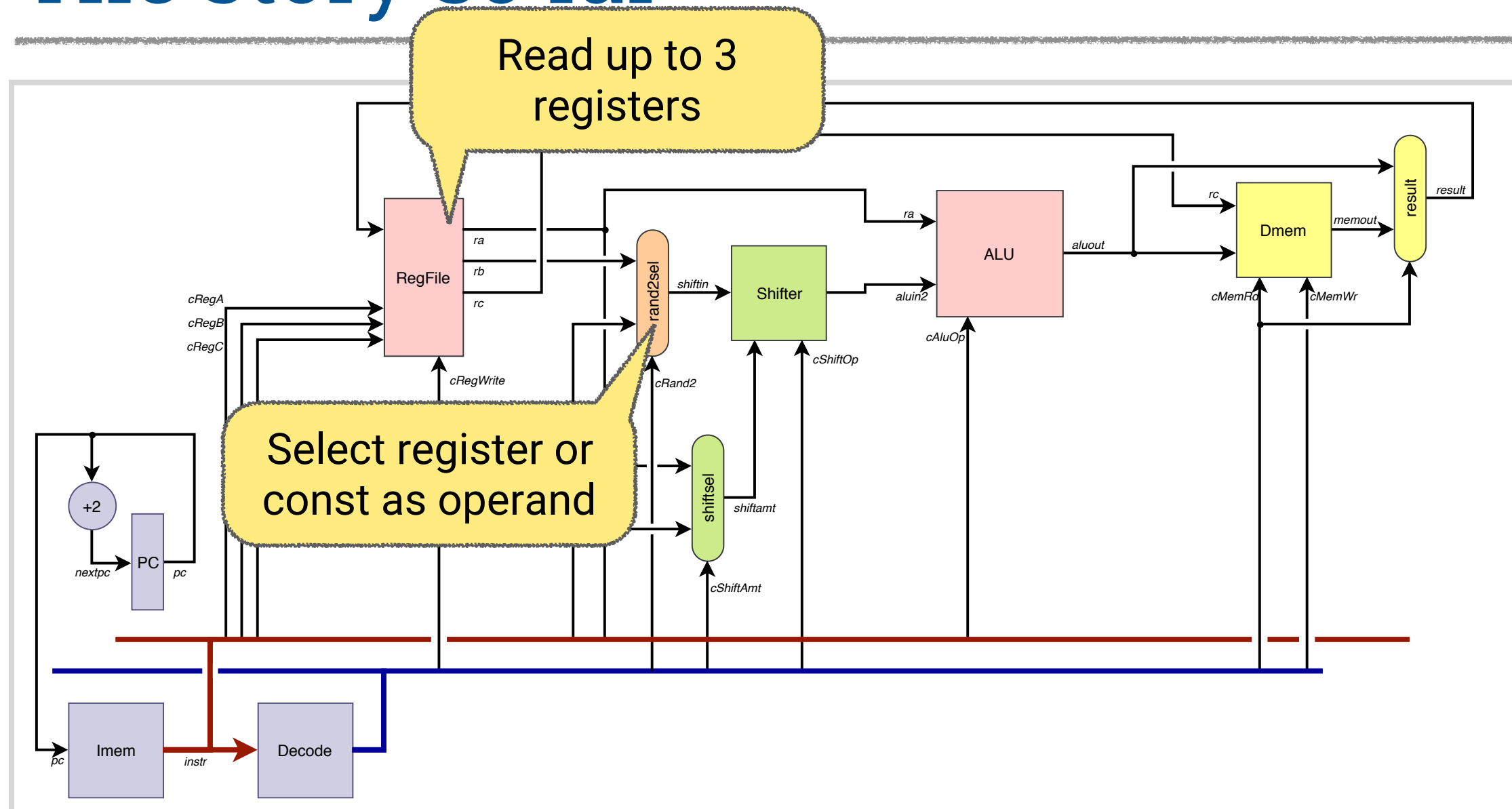
# The story so far



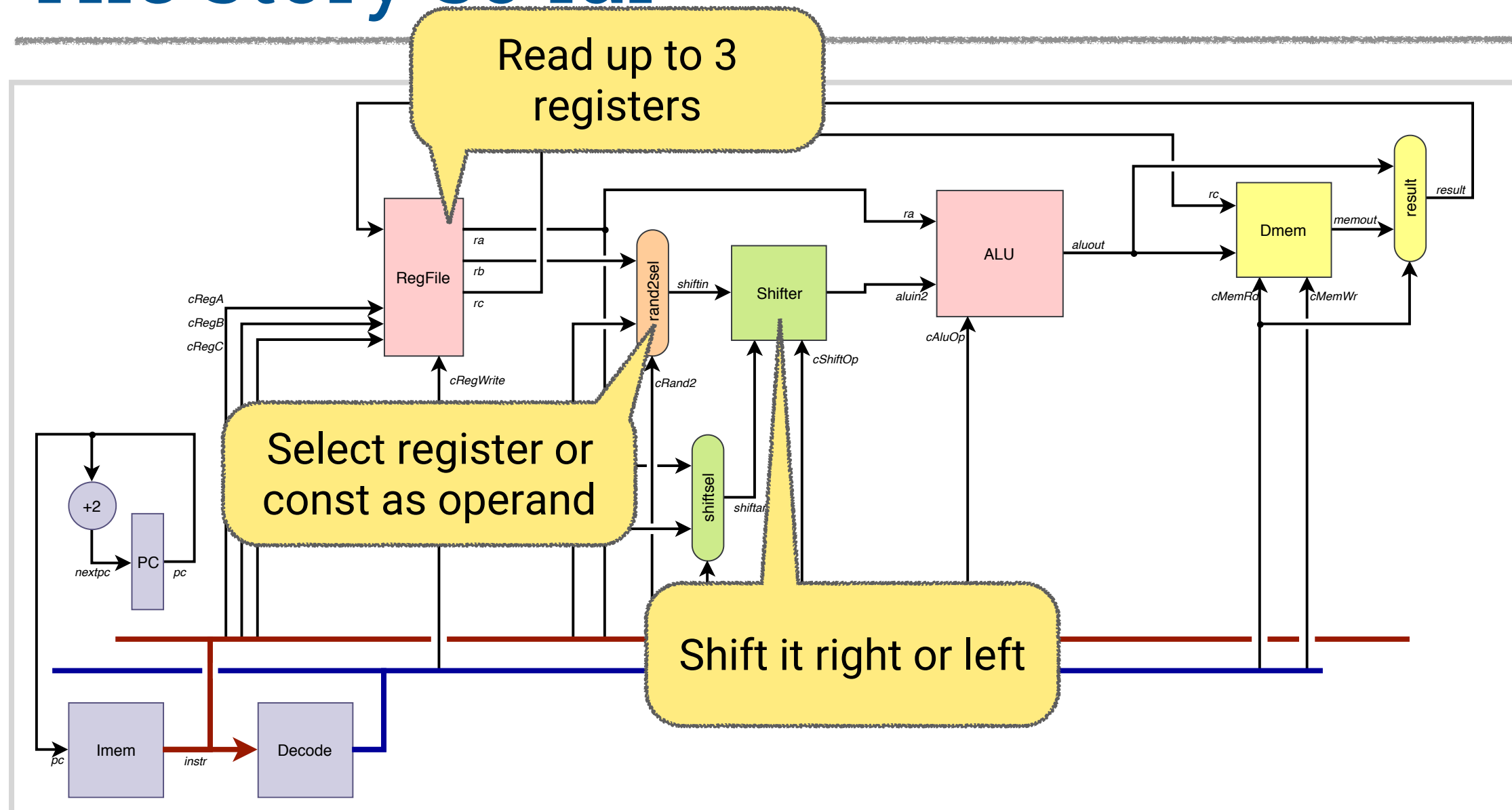
# The story so far



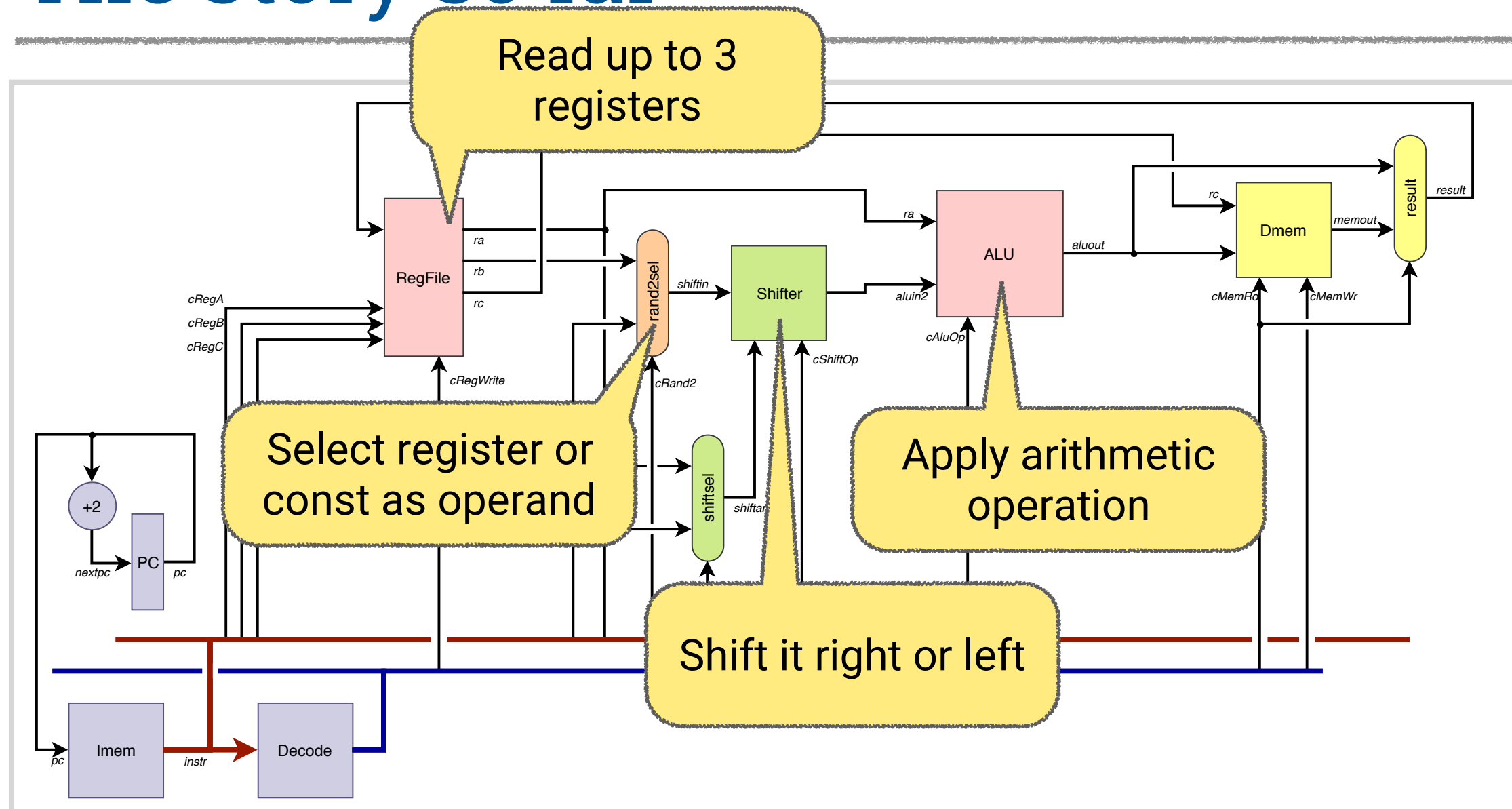
# The story so far



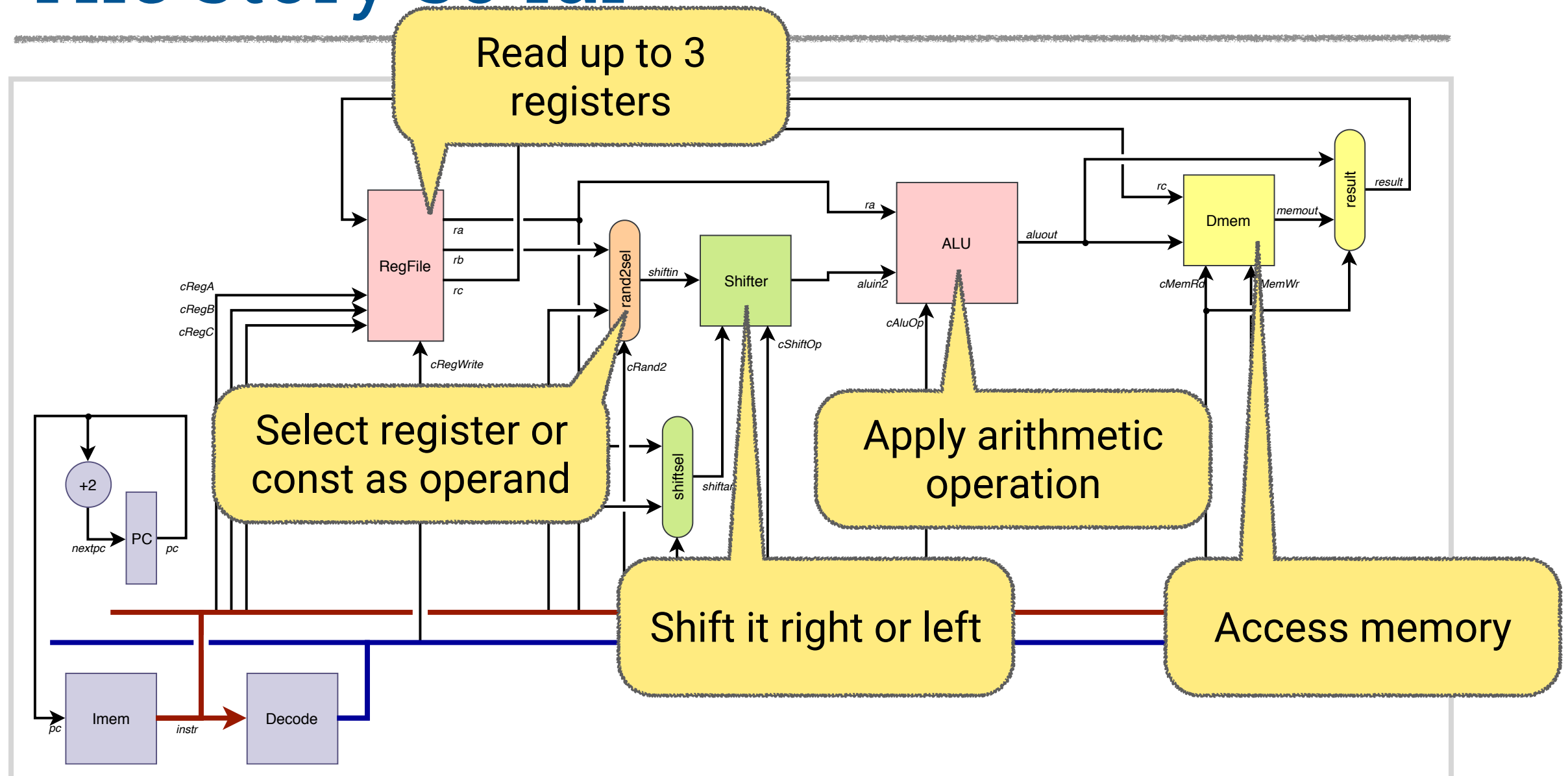
# The story so far



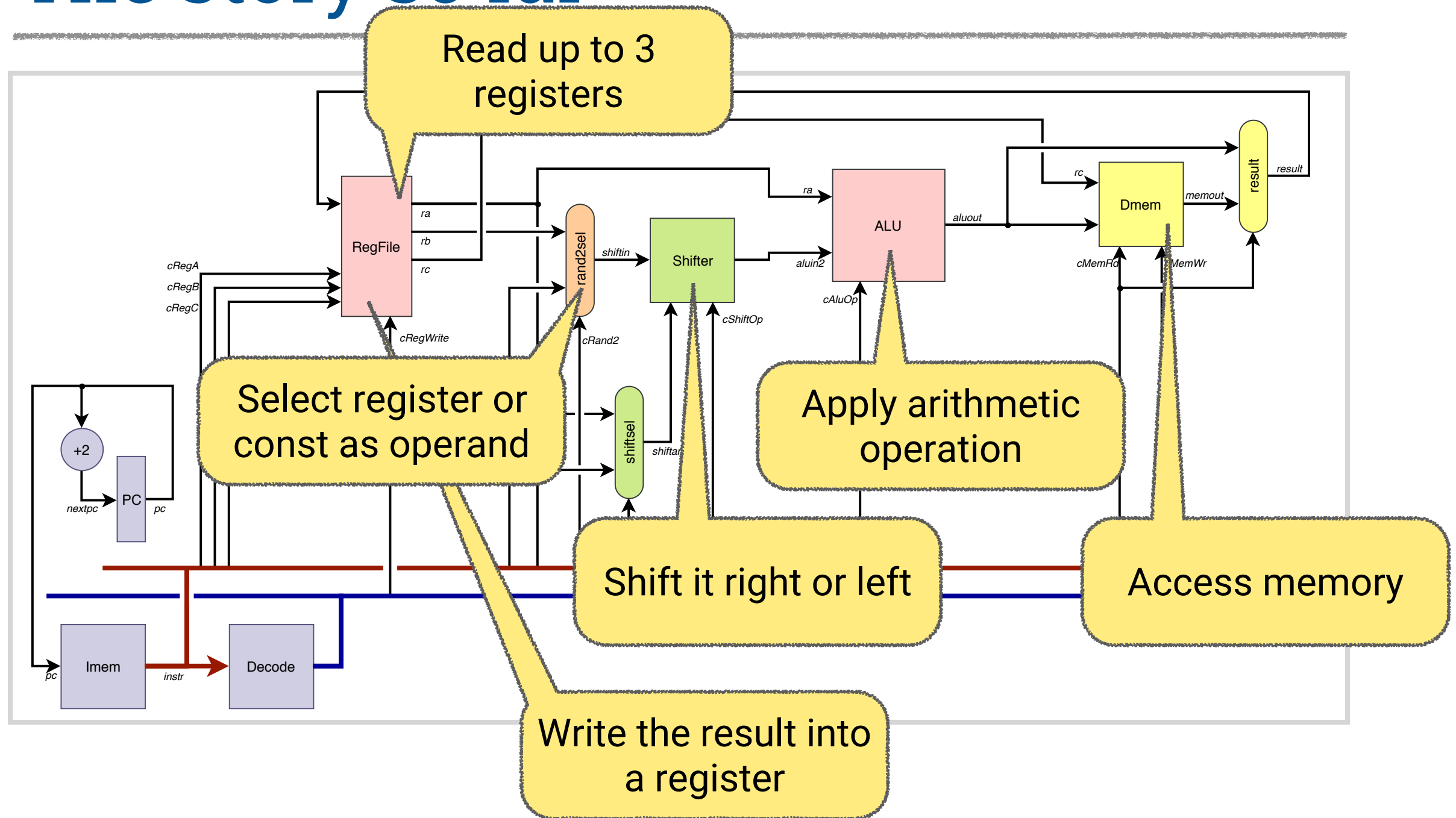
# The story so far



# The story so far



# The story so far



# Data and control

## Digital Systems – Lecture 22



Department of  
COMPUTER  
SCIENCE



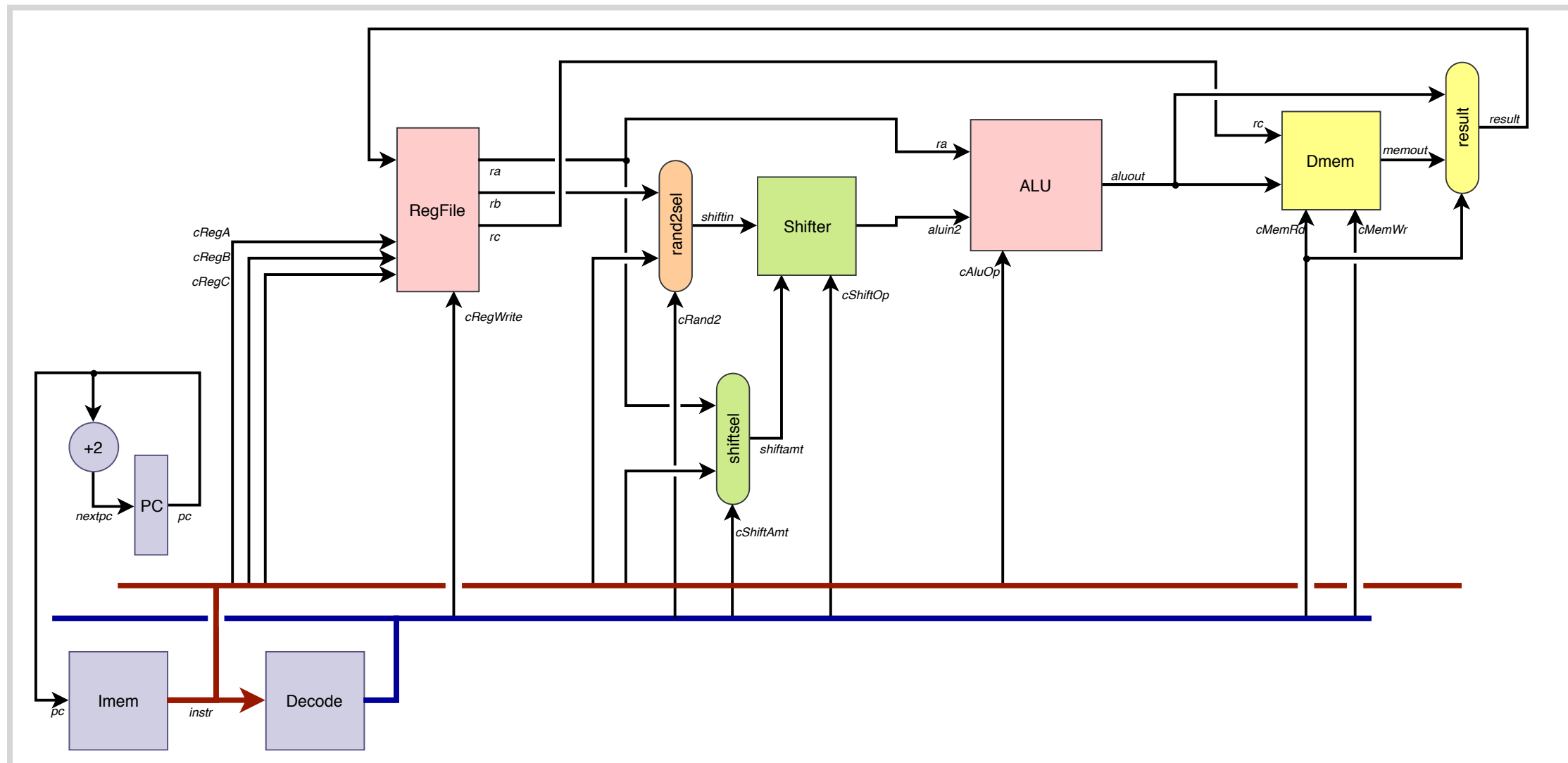
# In this lecture

---

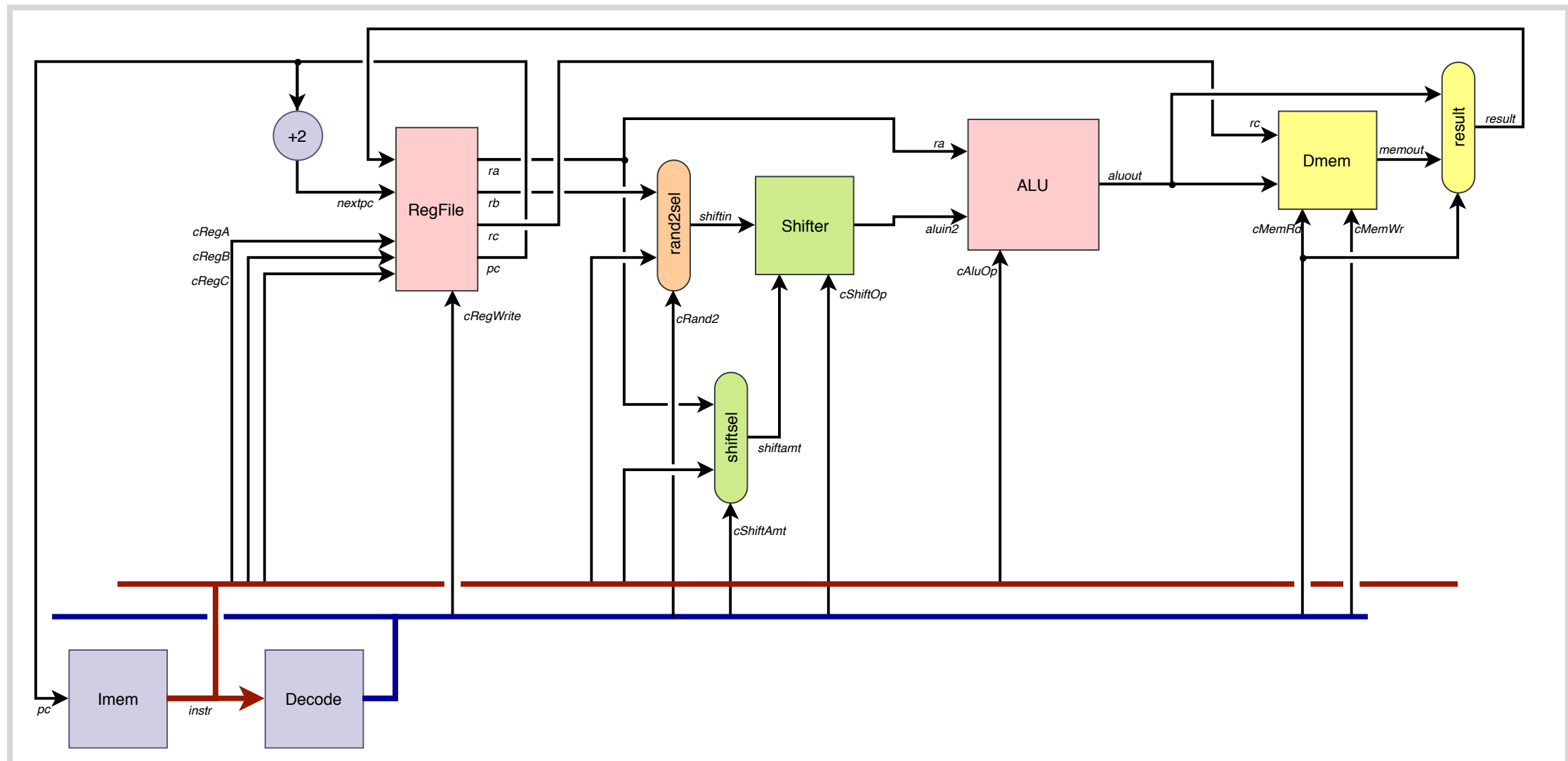
Further stages that enhance control elements of the datapath

- PC as a register
- Instruction decoding
- Subroutine calls
- Conditional execution

# The story from last time

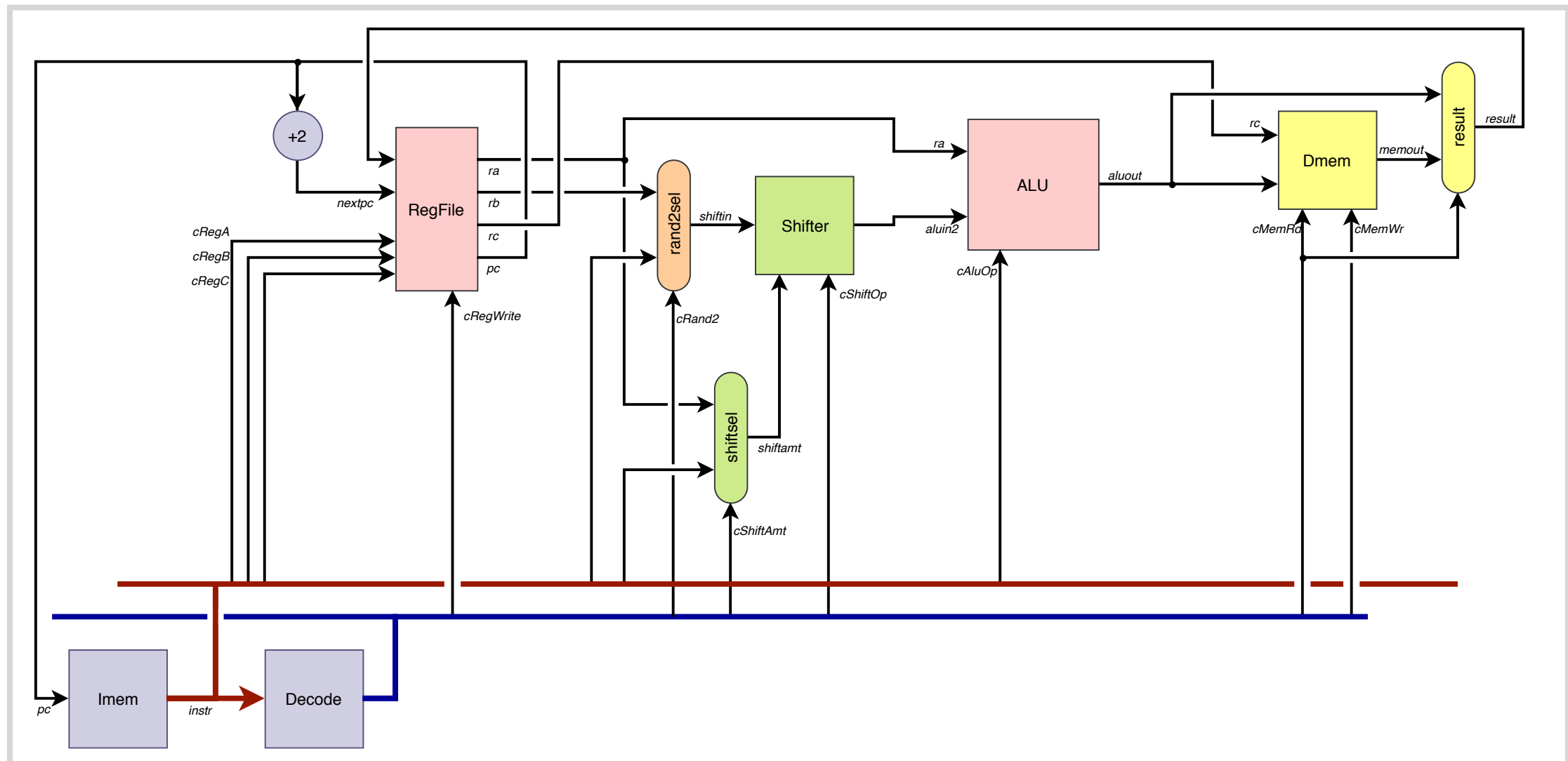


# Stage 6: PC as a register

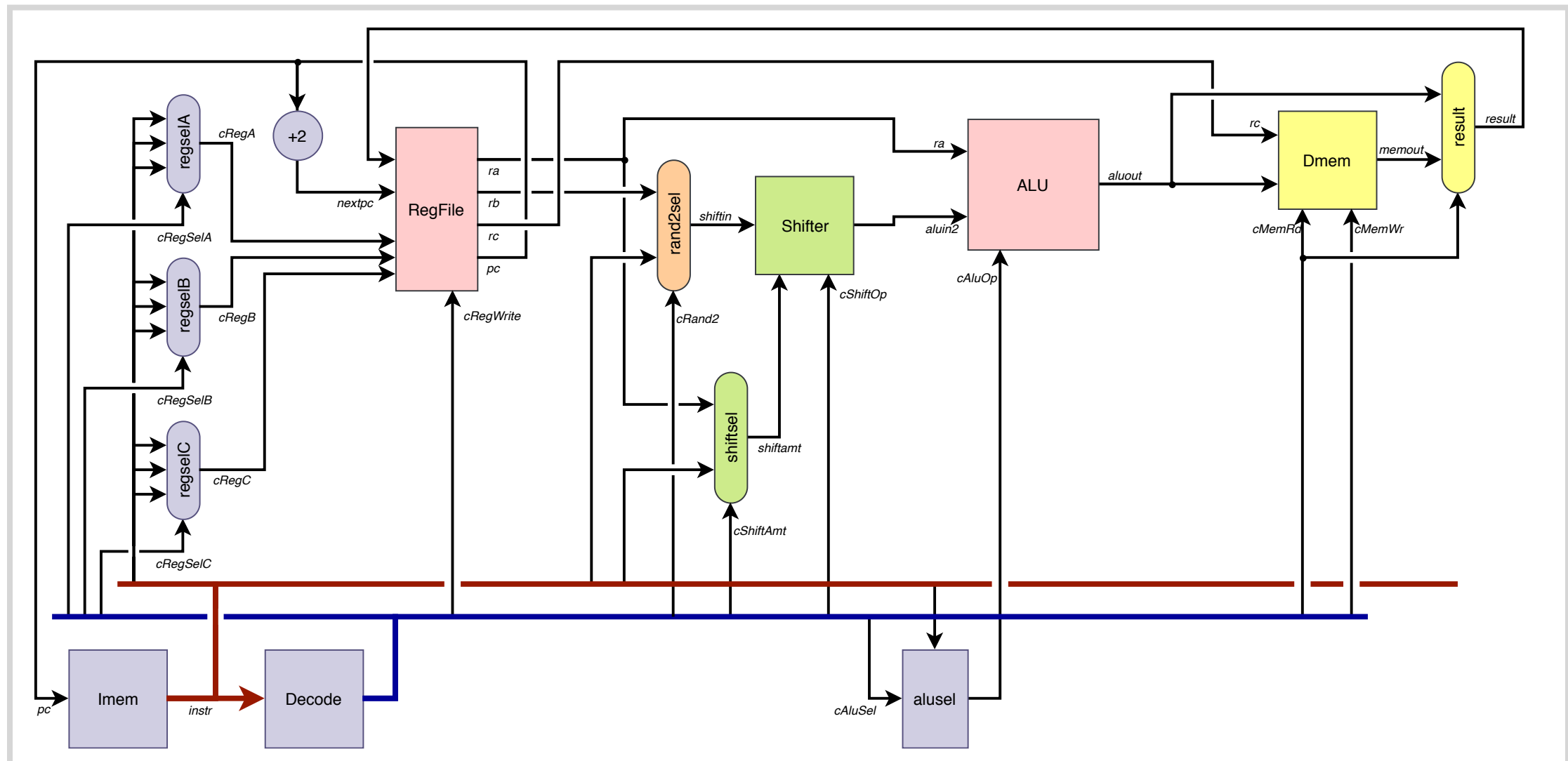




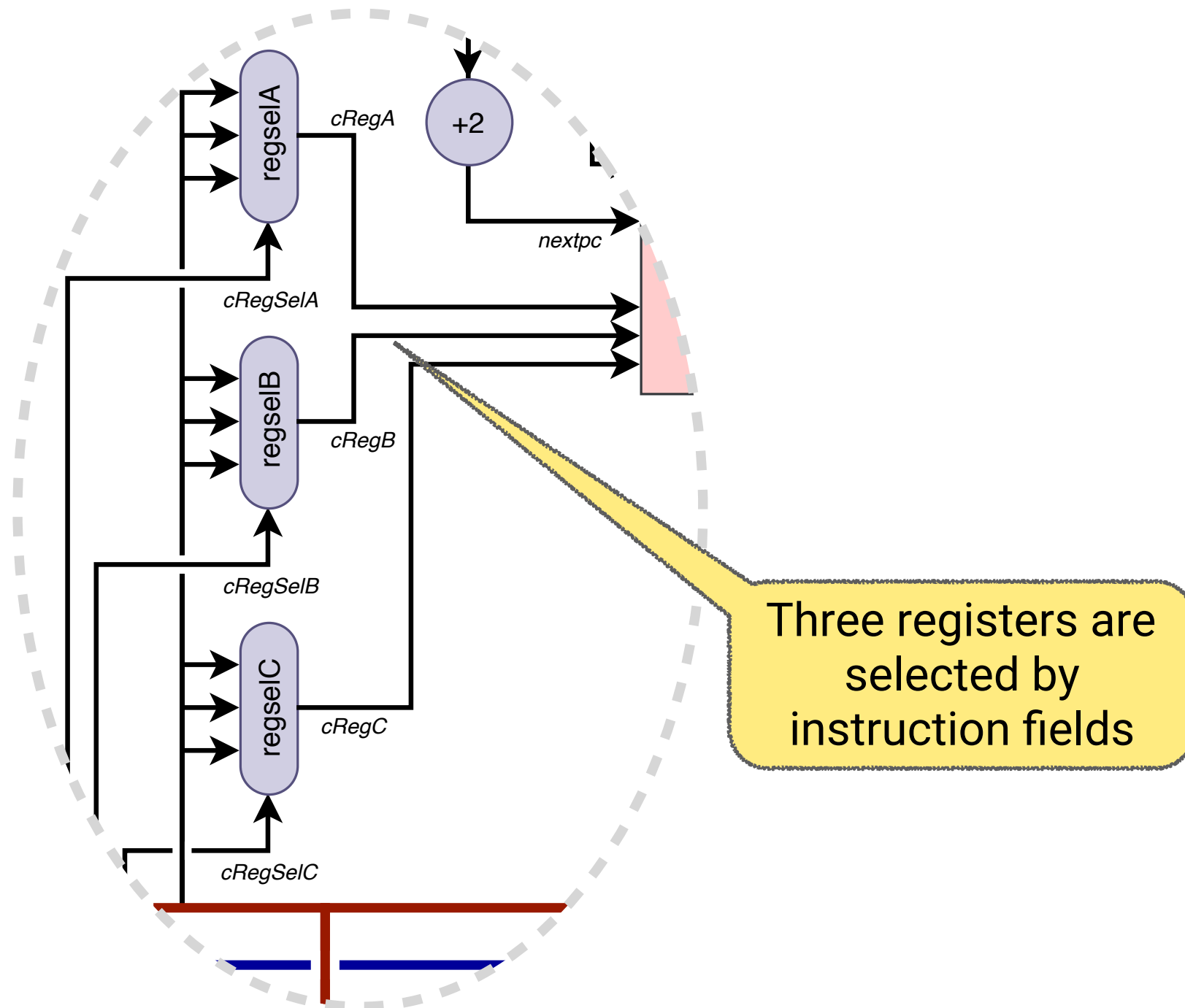
# Stage 6: PC as a register



# Stage 7: Instruction decoding



# Stage 7: Instruction decoding



# Controls for register selection

Instruction	cReg SelA	cReg SelB	cReg SelC	cRand2	cShiftOp/ Amt	cAlu Sel	cMem Rd/Wr	cReg Write
adds/subs r/i3	Ry	Rz	Rx	RImm3	Ls1/Sh0	Sg9	F/F	T
movs i8	—	—	Rw	Imm8	Ls1/Sh0	Mov	F/F	T
ldr r	Ry	Rz	Rx	RegB	Ls1/Sh0	Add	T/F	T
str r	Ry	Rz	Rx	RegB	Ls1/Sh0	Add	F/T	F
lsls i5	—	Ry	Rx	RegB	Ls1/ShImm	Mov	F/F	T
ands r	Rx	Ry	Rx	RegB	Ls1/Sh0	And	F/F	T
rors r	Ry	Rx	Rx	RegB	Ror/ShReg	Mov	F/F	T
ldr i5	Ry	—	Rx	Imm5	Ls1/Sh2	Add	T/F	T
str i5	Ry	—	Rx	Imm5	Ls1/Sh2	Add	F/T	F



# Instructions with fixed registers

add sp,sp,#⟨imm7⟩

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	0	0	0	0	0	imm7						

ldr ⟨Rw⟩,[sp,#⟨imm8⟩]

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	Rw			imm8							

sub sp,sp,#⟨imm7⟩

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	0	0	0	0	1	imm7						

b ⟨disp11⟩

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	disp11										

add ⟨Rw⟩,sp,#⟨imm8⟩

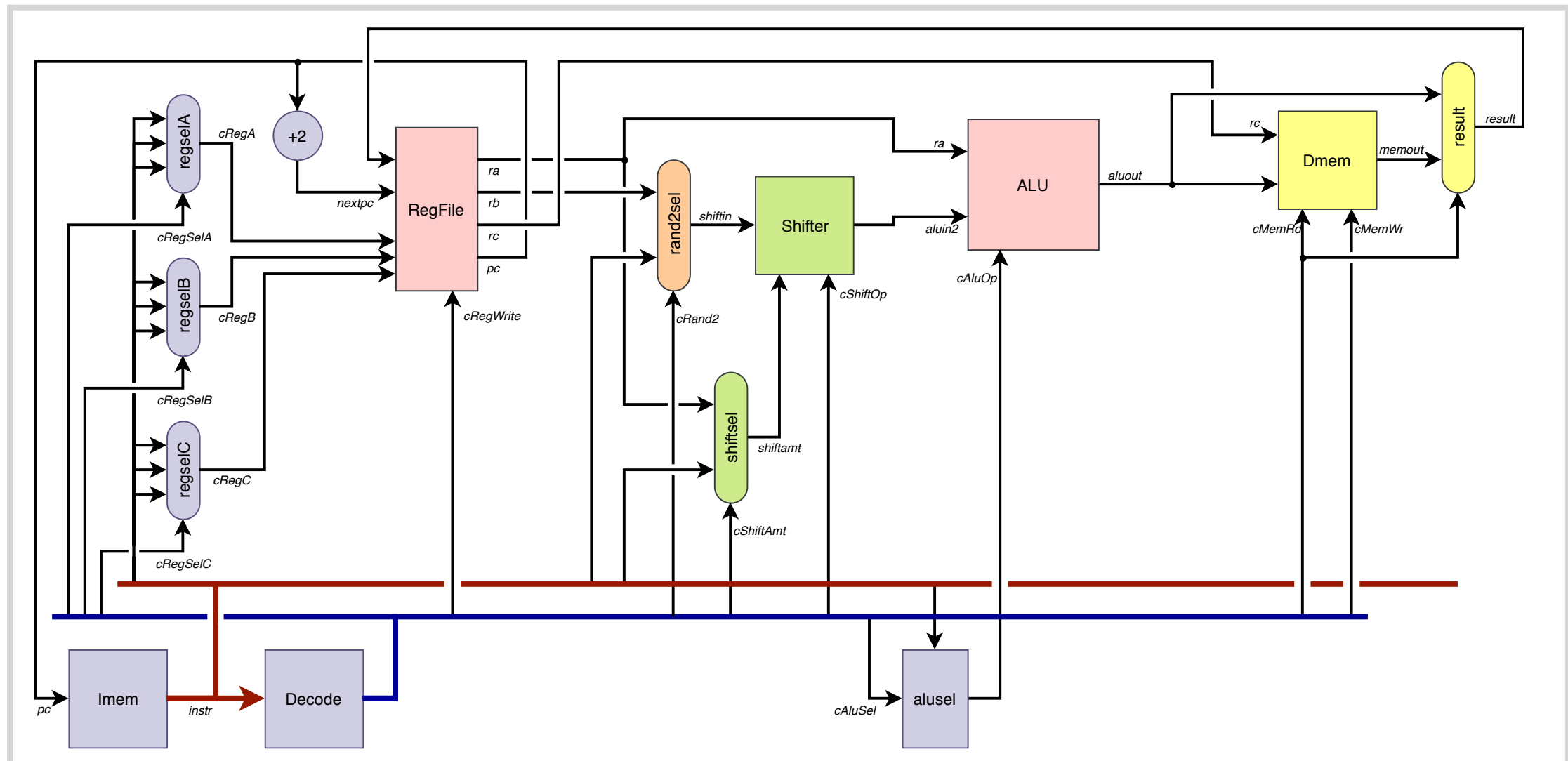
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	Rw			imm8							



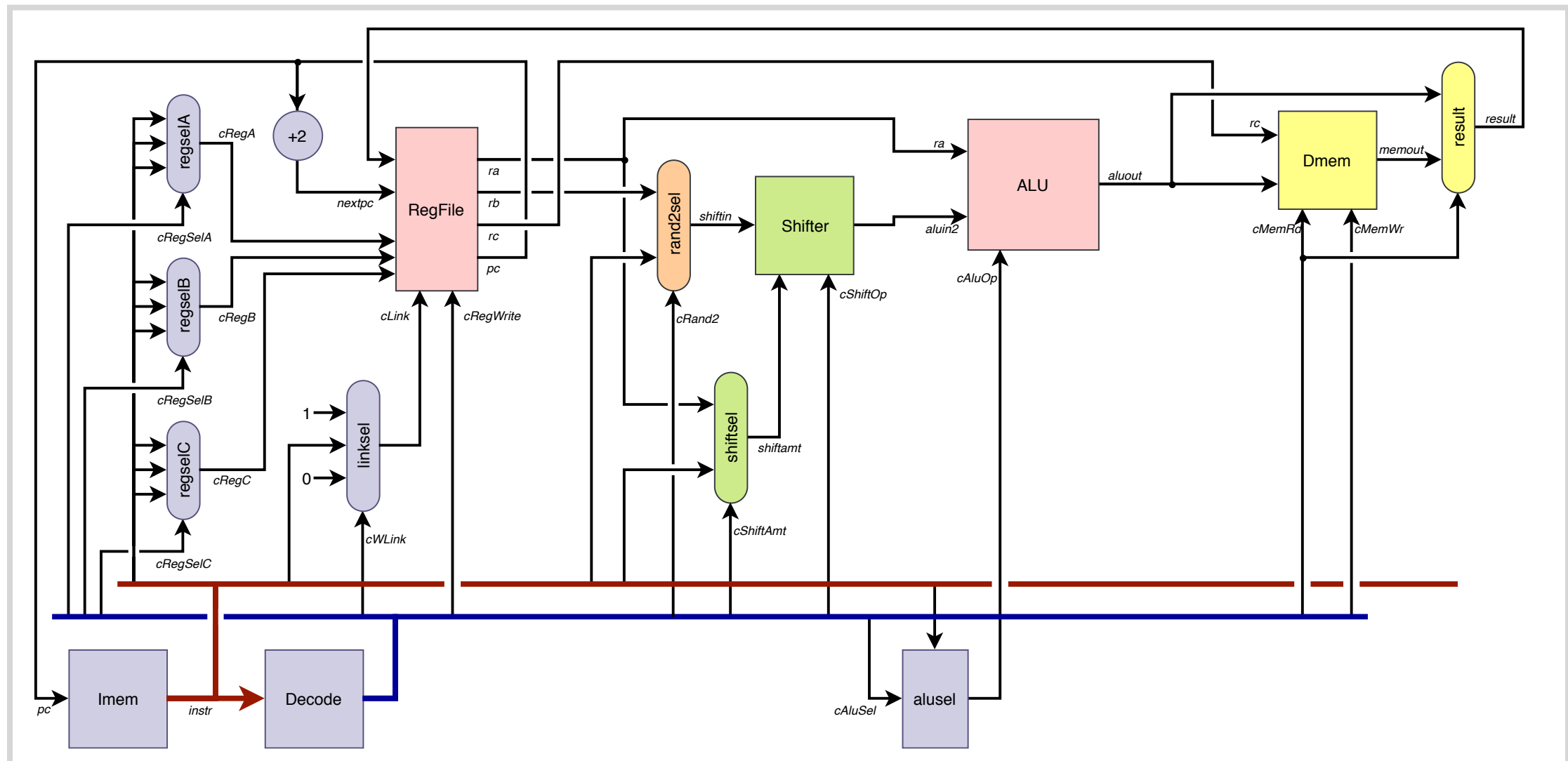
# More control rules

Instruction	cReg SelA	cReg SelB	cReg SelC	cRand2	cShiftOp/ Amt	cAlu Sel	cMem Rd/Wr	cReg Write
add/sub sp	Rsp	—	Rsp	Imm7	Ls1/Sh2	Sg7	F/F	T
add rsp	Rsp	—	Rw	Imm8	Ls1/Sh2	Add	F/F	T
ldr sp	Rsp	—	Rw	Imm8	Ls1/Sh2	Add	T/F	T
str sp	Rsp	—	Rw	Imm8	Ls1/Sh2	Add	F/T	F
b	Rpc	—	Rpc	SImm11	Ls1/Sh1	Add	F/F	T

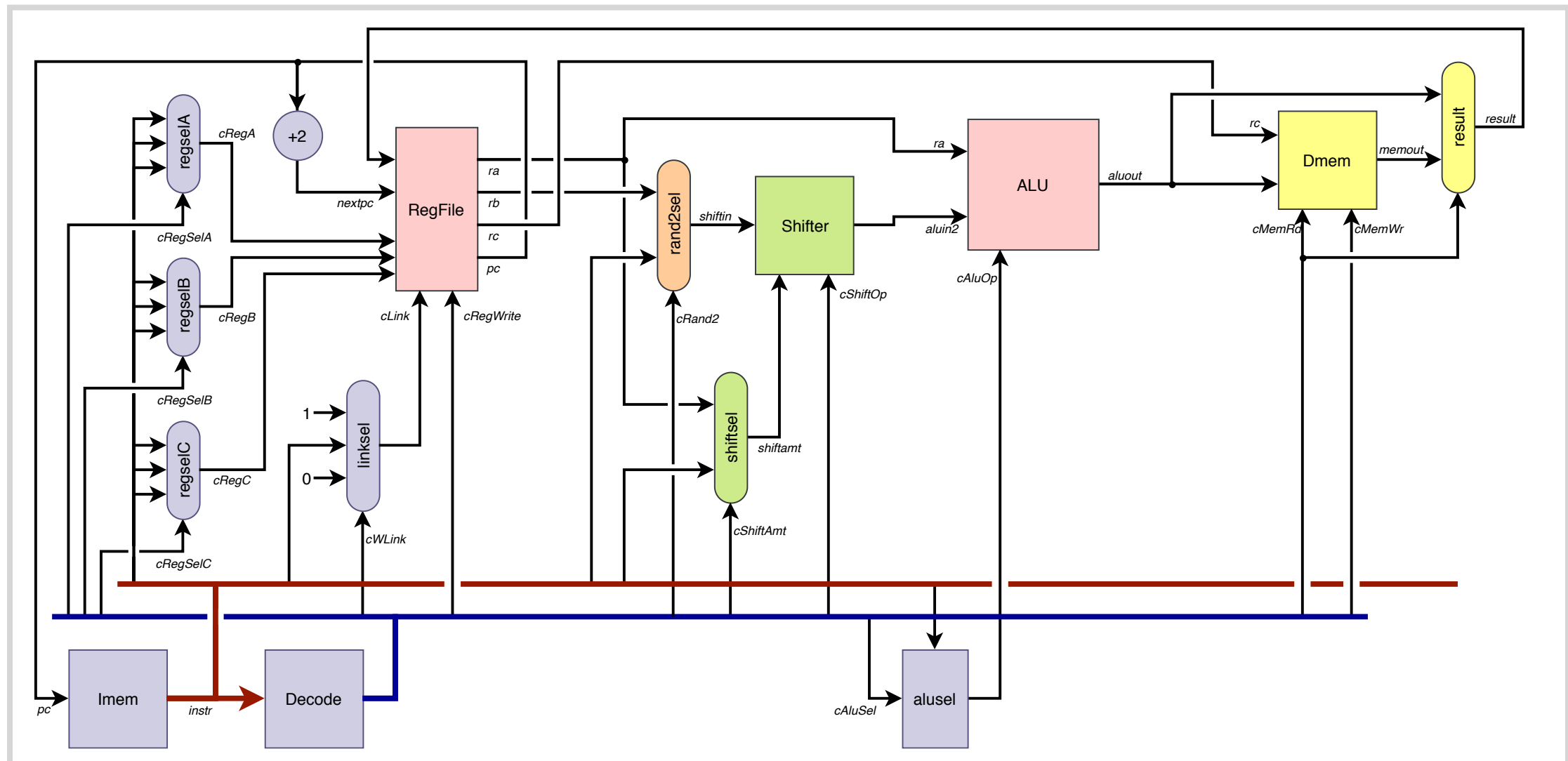
# Stage 7: Instruction decoding



# Stage 8: Subroutine calls



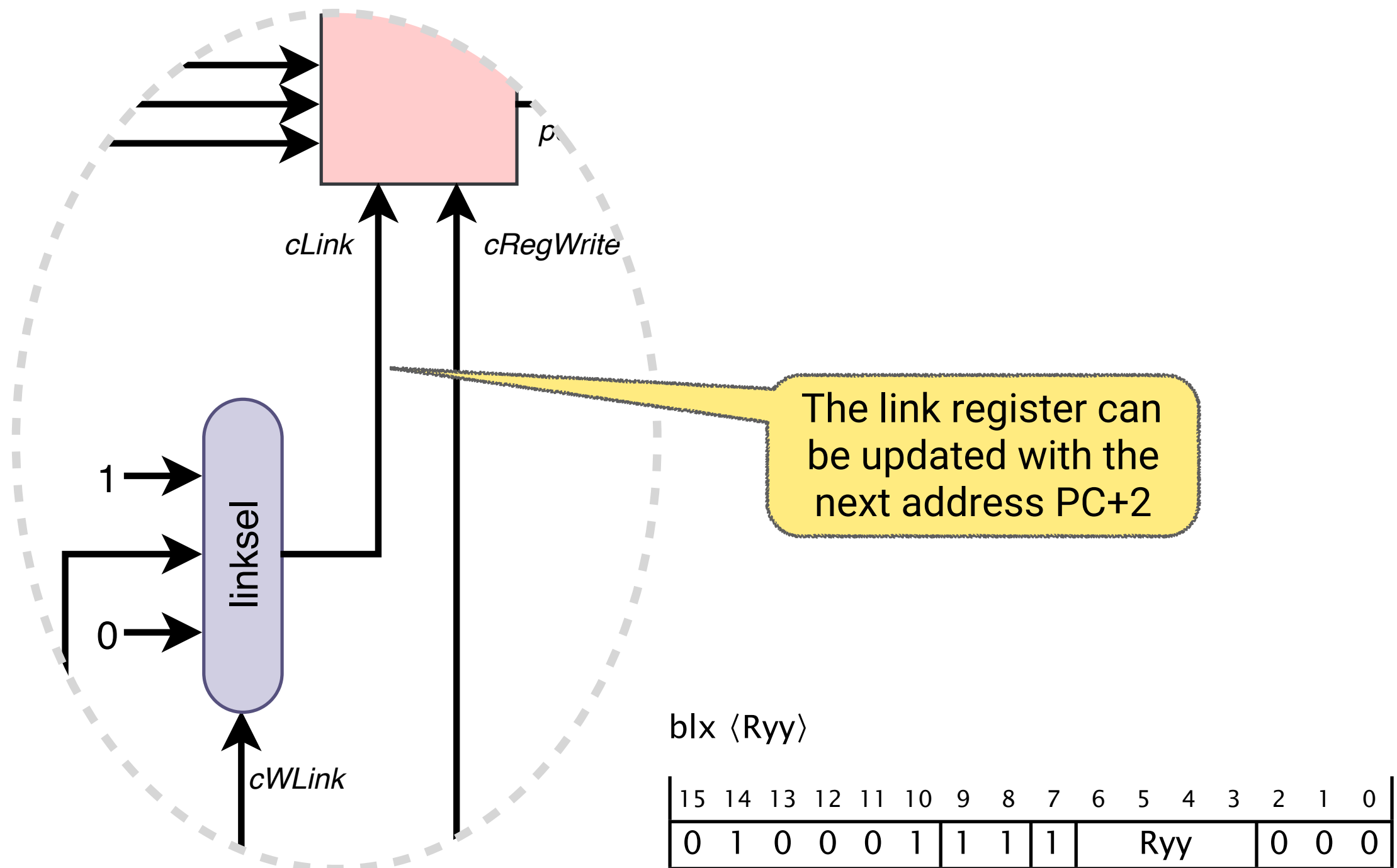
# Stage 8: Subroutine calls



blx <Ryy>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	1	1	1	Ryy				0	0	0

# Stage 8: Subroutine calls



# Subroutine call and return

bx <Ryy>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	1	1	0	Ryy			0	0	0	

blx <Ryy>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	1	1	1	Ryy				0	0	0

Instruction	cRegSel A/B/C	cRand2	cShiftOp/ Amt	cAlu Sel	cMem Rd/Wr	cReg Write	cWLink
bx/blx r	−/Ryy/Rpc	RegB	Ls1/Sh0	Mov	F/F	T	C

# B1 done the old-fashioned way

b11,  $\langle \text{simm10} \rangle$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	S	imm10									

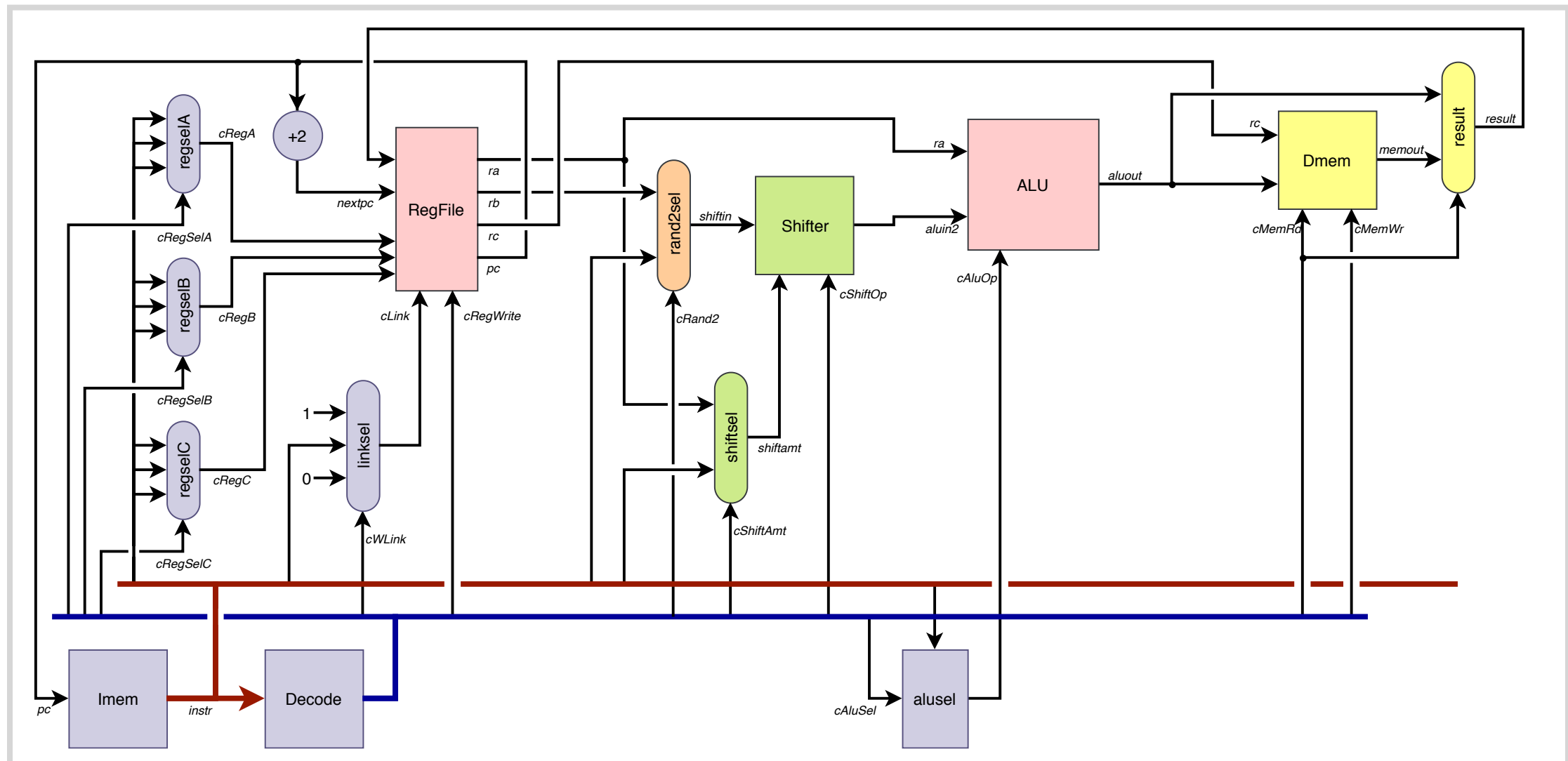
b12,  $\langle \text{imm11} \rangle$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	J1	1	J2	imm11										

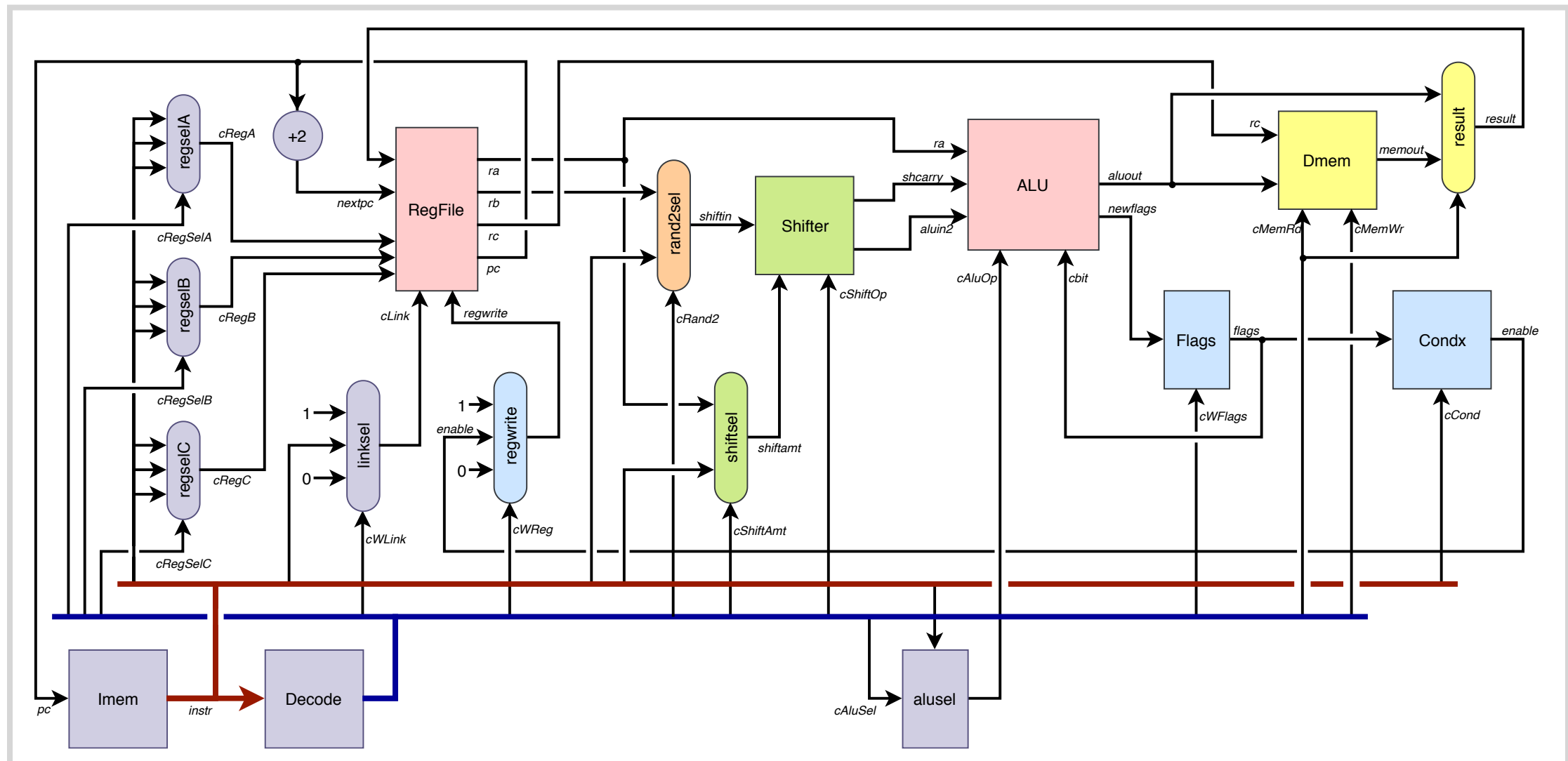
Instruction	cRegSel A/B/C	cRand2	cShiftOp/ Amt	cAlu Sel	cMem Rd/Wr	cReg Write	cWLink
b11	Rpc / - / R1r	SImm11	Ls1/Sh12	Add	F / F	T	N
b12	R1r / - / Rpc	Imm11	Ls1/Sh1	Add	F / F	T	Y



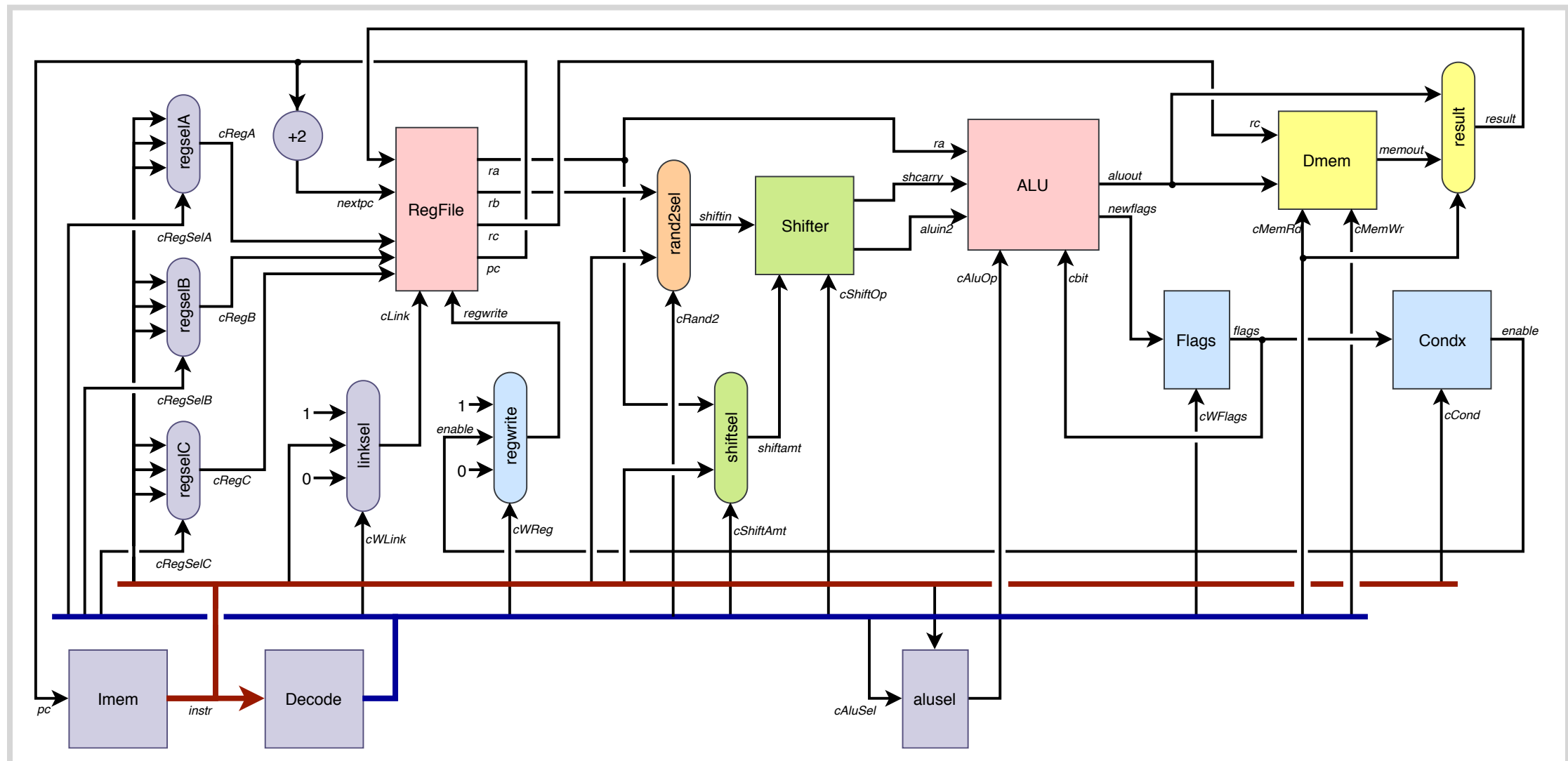
# Stage 8: Subroutine calls



# Stage 9: Conditional execution



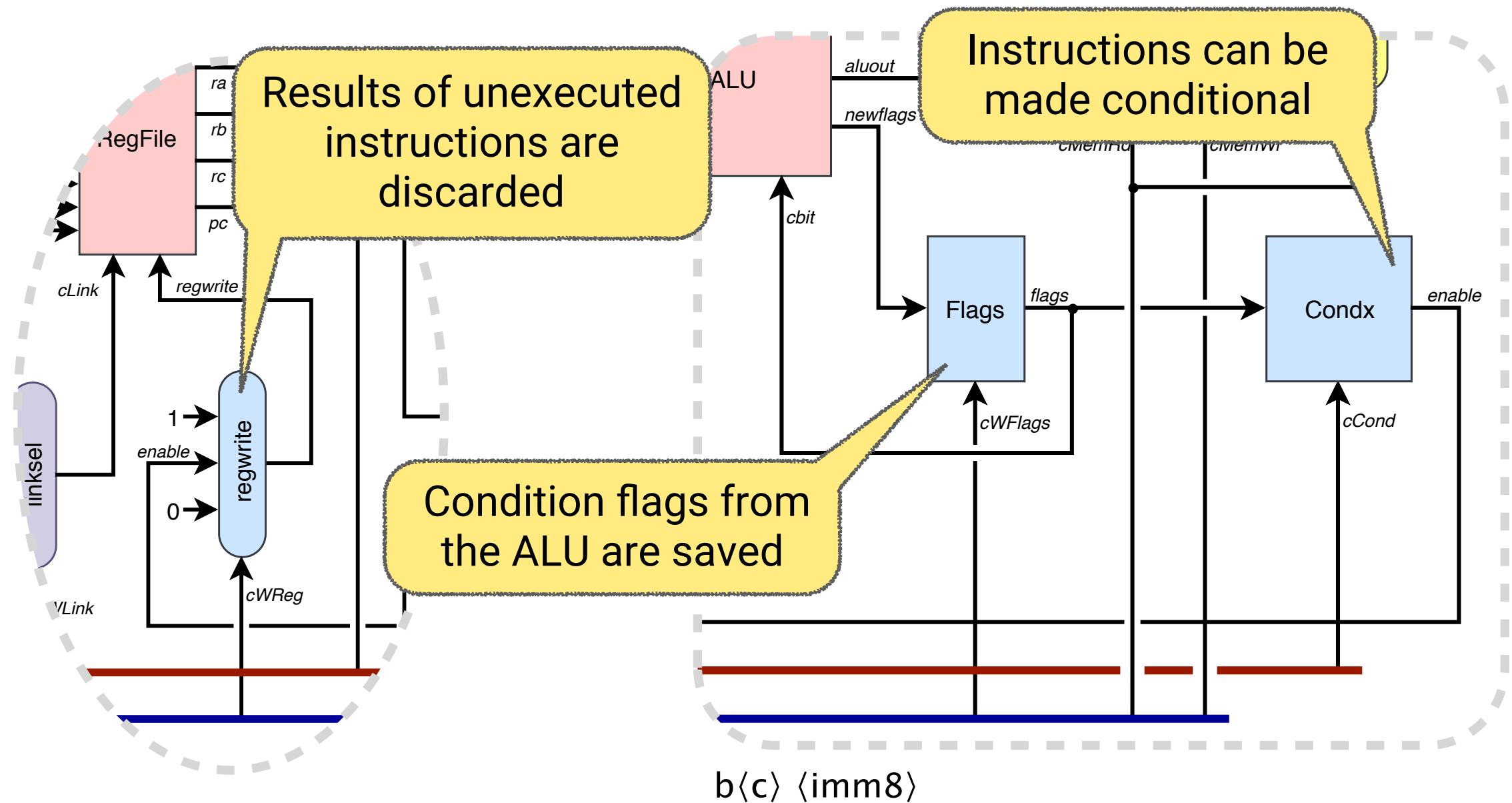
# Stage 9: Conditional execution



b(c) <imm8>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	cond				imm8							

# Stage 9: Conditional execution



# Compare and branch

b⟨c⟩ ⟨imm8⟩

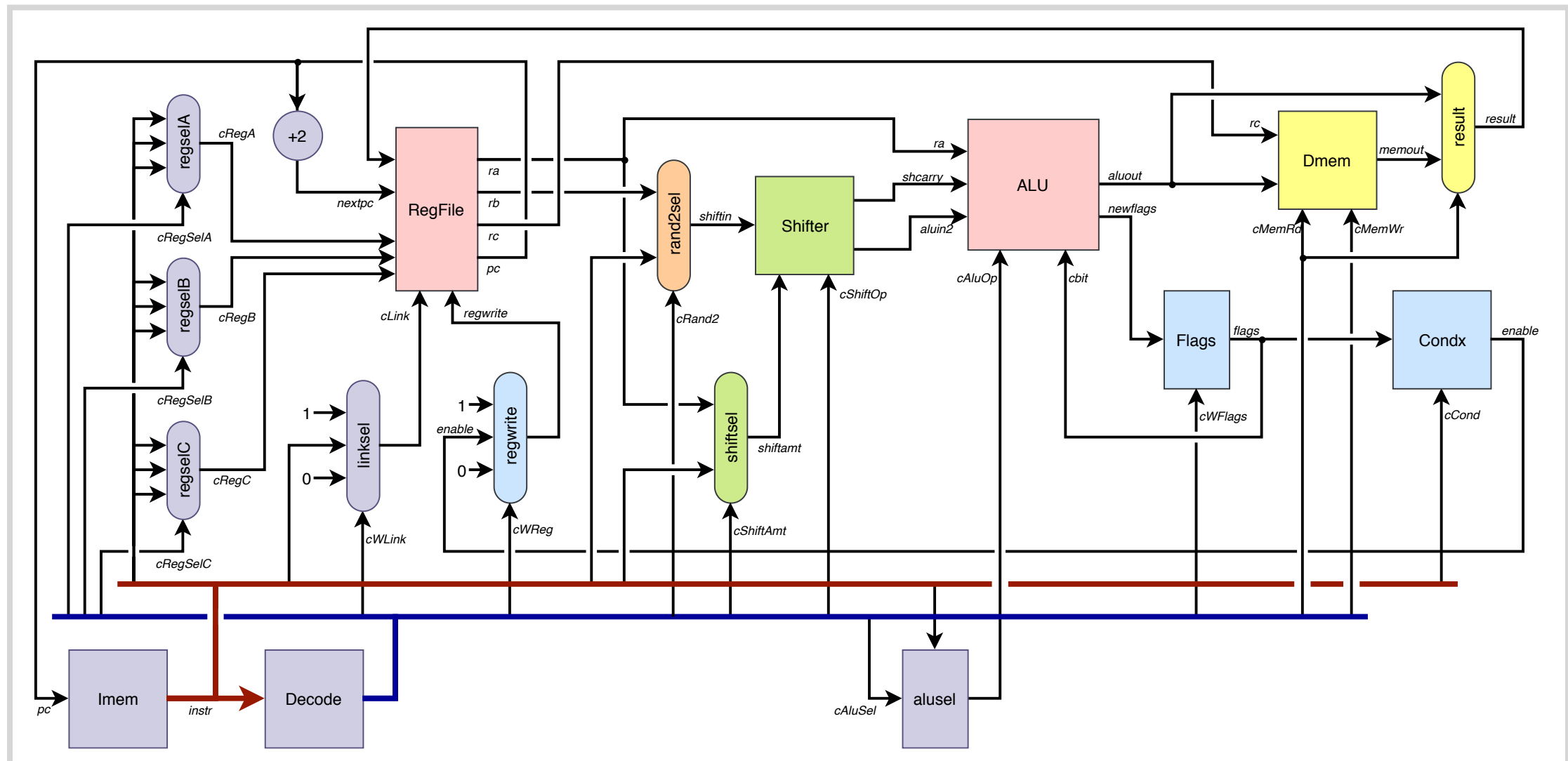
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	cond				imm8							

cmp ⟨Rw⟩, #⟨imm8⟩

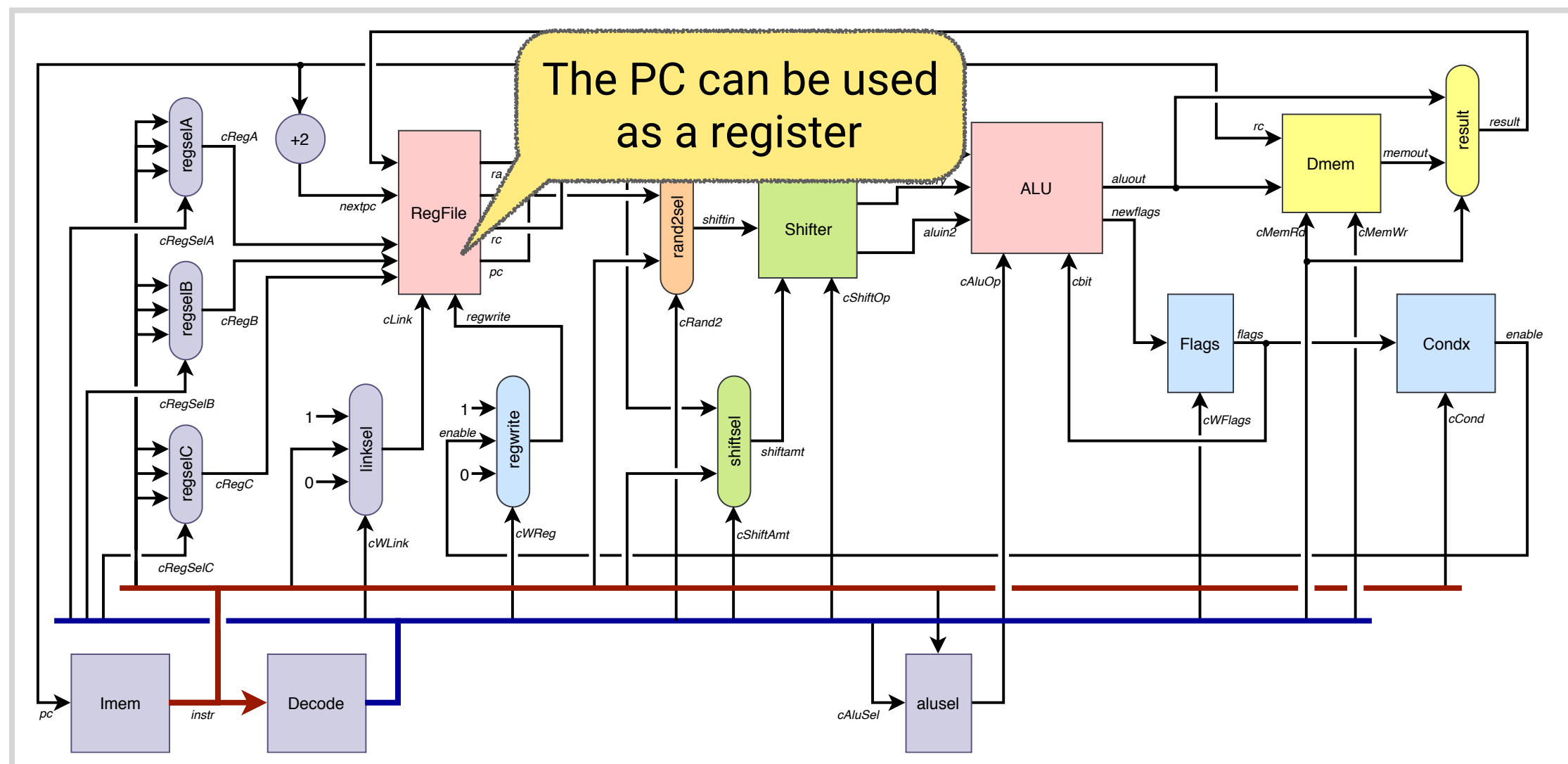
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	Rw			imm8							

Instruction	cRegSel A/B/C	cRand2	cShiftOp/ Amt	cAlu Sel	cMem Rd/Wr	cWFlags/ Reg/Link
b<c>	Rpc / – / Rpc	SImm8	Ls1/Sh1	Add	F / F	F / C / N
cmp ri	Rw / – / –	Imm8	Ls1/Sh0	Sub	F / F	T / N / N
subs ri	Rw / – / Rw	Imm8	Ls1/Sh0	Sub	F / F	T / Y / N

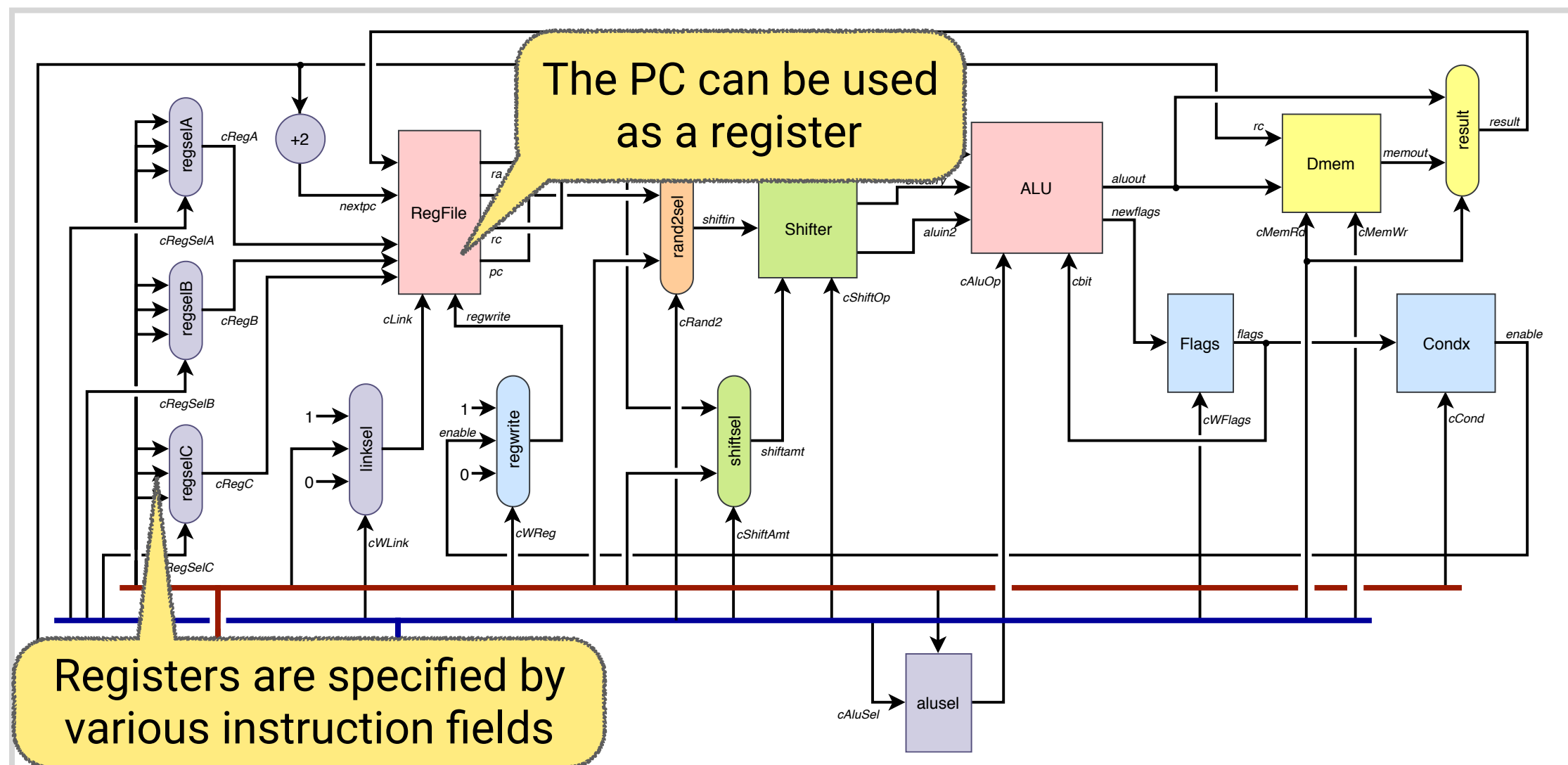
# The complete datapath



# The complete datapath

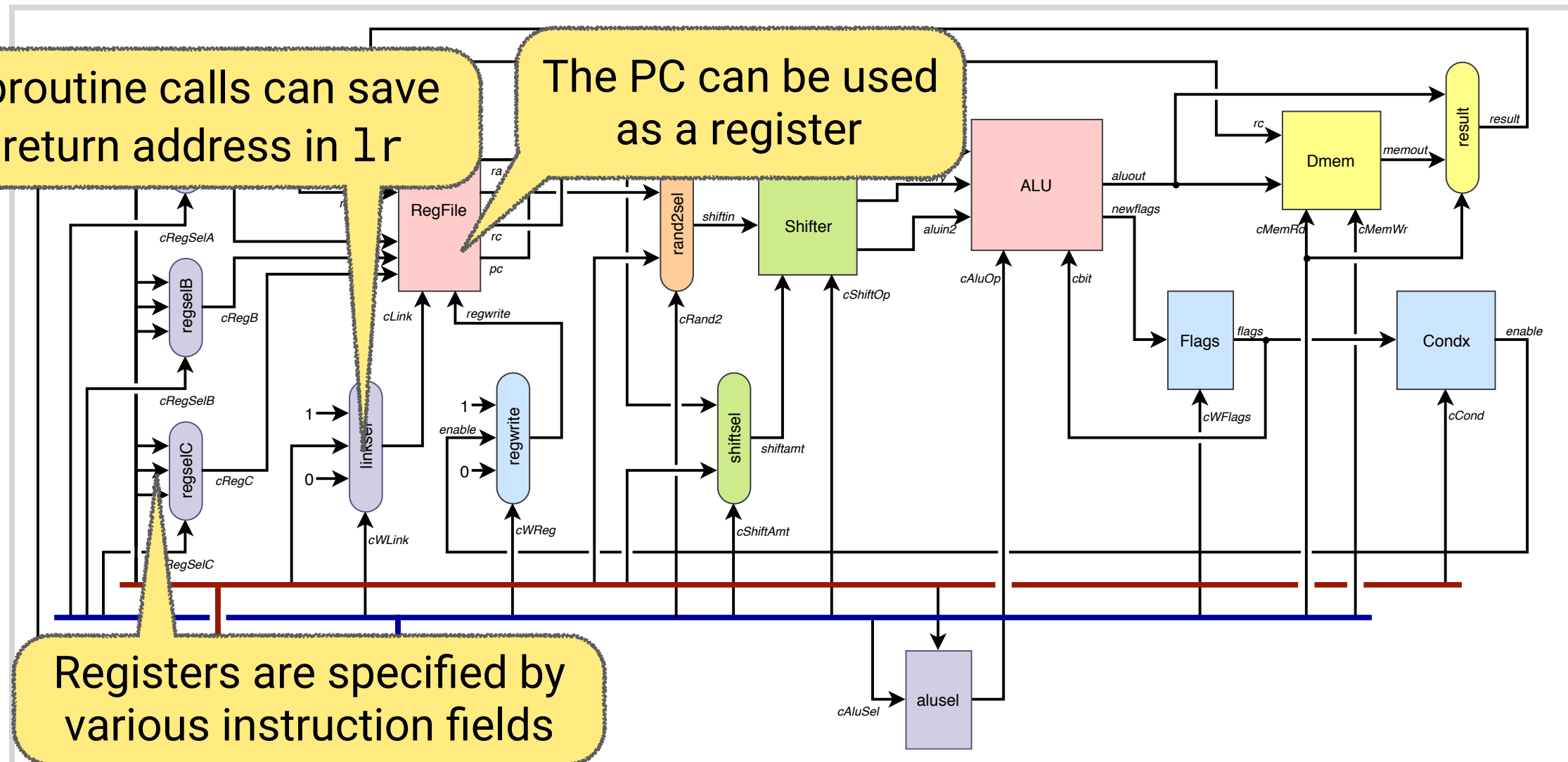


# The complete datapath

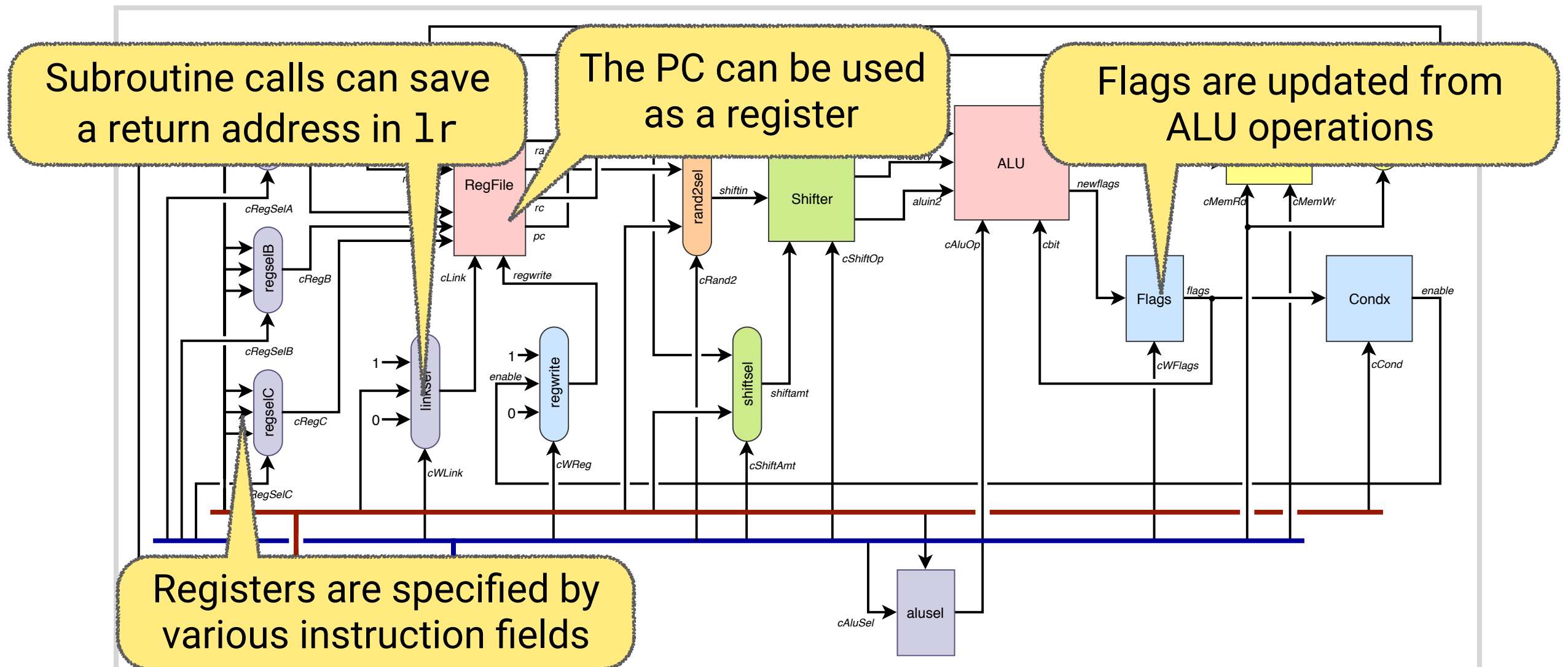




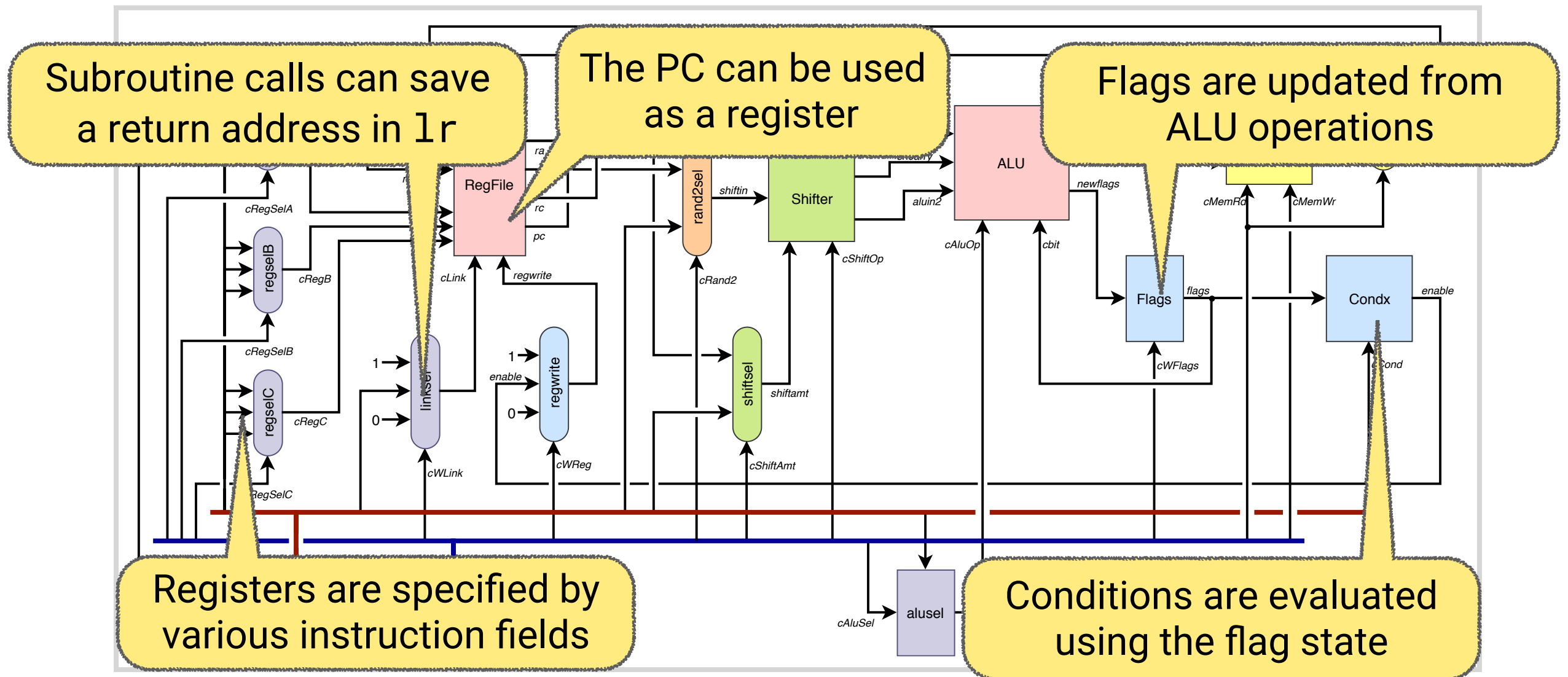
# The complete datapath



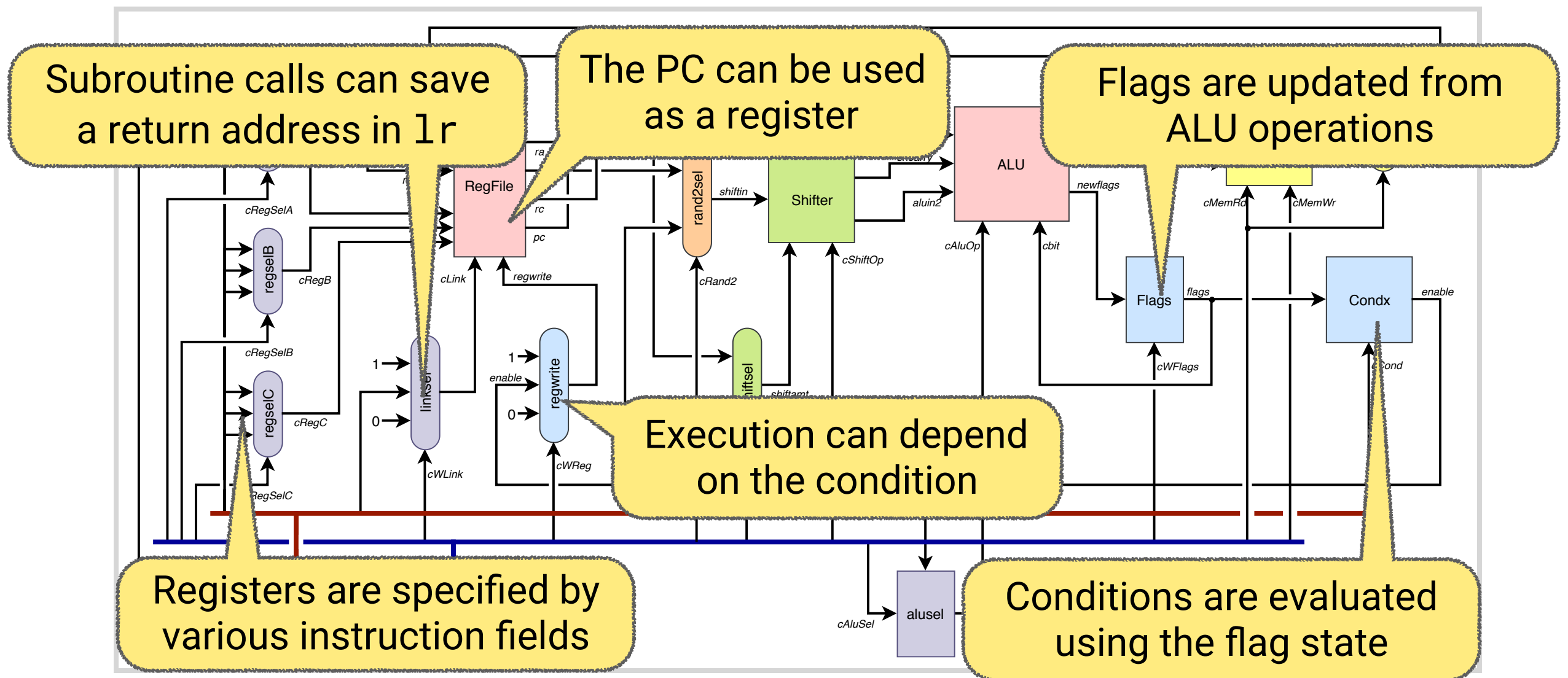
# The complete datapath



# The complete datapath



# The complete datapath



# A simulator

---

“Lab 5” consists of a simulator following the design given in these lectures.

It can load and execute programs prepared with the ARM assembler and linker, provided they use no instructions that we haven’t implemented.

A document on the wiki presents the program and explanation in ‘literate’ form.